SCREENING HEURISTICS FOR THE EVALUATION OF COVERT NETWORK NODE INSERTION SCENARIOS

THESIS

Andrew Pekarek, Major, USA

AFIT-ENS-MS-22-M-161

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-22-M-161

SCREENING HEURISTICS FOR THE EVALUATION OF COVERT NETWORK NODE INSERTION SCENARIOS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Andrew E. Pekarek, BS, MSCM

Major, USA

March 2022

AFIT-ENS-MS-22-M-161

SCREENING HEURISTICS FOR THE EVALUATION OF COVERT NETWORK NODE INSERTION SCENARIOS

Andrew E. Pekarek, BS, MSCM

Major, USA

Committee Membership:

Maj Michael J. Garee, PhD
Chair

Dr. Nathan B. Gaw
Member

Abstract

The majority of research on covert networks uses social network analysis (SNA) to determine critical members (leaders, technical experts, or key operatives) of the network to either kill or capture for the purpose of network destabilization. This thesis takes the opposite approach and evaluates potential scenarios for inserting an agent into a covert network for information gathering purposes or future disruption operations. An insertion scenario consists of one or more members of the network with whom an inserted agent targets for relationship development. The agent leverages these relationships to gain the desired effect on the network. Due to the substantial number of potential insertion scenarios in a large network, evaluating all possible scenarios can require hours of computational time. This research proposes three screening heuristics that leverage SNA measures to reduce the solution space before applying a simple search heuristic. Each heuristic is tested on networks of various sizes and types for accuracy and time and compares these results to a "brute force" all-scenarios model. All presented heuristics reduce the computational time by 99%, with one heuristic averaging a 0.90 accuracy across all test networks.

To my wife and fur-kids

Acknowledgements

I would like to thank my wife for her patience and support throughout the entire thesis process. Thank you to my advisor for keeping me on track and getting me to the end. Finally, thank you to my dogs and cat for keeping me company during the late nights.

Andrew E. Pekarek

Table of Contents

List of Figures

List of Tables

SCREENING HEURISTICS FOR THE EVALUATION OF COVERT NETWORK NODE INSERTION SCENARIOS

I. Introduction

1.1 Background

The *2021 Interim National Security Strategic Guidance* (INSSG) and *The Department of Homeland Security Strategic Plan: Fiscal Years 2020-2024* identify the international and domestic threats to the United States (US). These threats fall into three categories: near-peer competitors, regional actors, and non-state actors. China and Russia are the near-peer competitors, aggressively asserting themselves on the world stage to gain global influence. They are the primary focus of the national security strategy. Regional actors such as Iran and North Korea seek to disrupt regional stability, pursue nuclear capabilities, and threaten the US and its allies. The final category − and the focus of this research − are the transnational non-state actors that seek the attack the US and undermine the American way of life, and they take the form of terrorist, violent extremist, and transnational criminal organizations [1], [2].

All three types of non-state actors operate in the shadows of society, continually evolving and leveraging technology to conceal their operations and members. Terrorist and violent extremist organizations exploit the anonymity of the Internet to spread their ideology and propaganda worldwide, radicalizing new recruits to violence [2]. Criminal organizations have also leveraged the Internet to further their reach as they pursue criminal activities that "include trafficking and smuggling of humans, drugs, weapons,

and wildlife, as well as money laundering, corruption, cybercrime, fraud, and financial crimes" [2]. Due to their desire to remain hidden, terrorist, extremist, and criminal organizations are also referred to as either dark, clandestine, or covert organizations.

To counter the actions of these covert organizations, specifically terrorist and violent extremist, the INSSG directs a *whole of government* approach, utilizing the "full array of tools" to include law enforcement and intelligence capabilities, while ensuring information sharing among agencies [1]. For the Department of Homeland Security, actionable intelligence is the basis for their strategy to counter terrorist, extremist, and criminal organizations:

> *Effective homeland security operations rely on timely and actionable intelligence to accurately assess and prevent threats against the United States. Accordingly, DHS works diligently to enhance intelligence collection, integration, analysis, and information sharing capabilities to ensure partners, stakeholders, and senior leaders receive actionable intelligence and information necessary to inform their decisions and operations* [2].

One tool being utilized by the US government to perform analysis of and to counter covert organizations is *social network analysis* (SNA). Like societies and other organizations, covert organizations consist of a network of people tied together by relationships, ideas, and goals. SNA leverages the network structure and utilizes elements of graph theory to draw key insights about the organization [3].

Covert networks present unique challenges to analysts, as these networks do not follow certain norms seen in typical social networks. The need for secrecy and survival affects how the network's structure develops. For example, the structure of terrorist

networks is usually cell-centric. The greatest density of relationships is within the cells, with usually only one connection outside of the cell to other parts of the network. The relationships or connections that do exist within the network are difficult to detect as the members are focused on hiding those connections. Due to the emphasis placed on hiding the network, the lack of information about the network further challenges analysis of the network [4].

Previous studies of covert networks focused on node deletion as a means of causing disruption in covert networks through the targeting, capturing, or killing of key members within the network [5]. The opposite approach of node deletion is node insertion. Node insertion entails inserting an agent or sensor into a network with a goal of establishing relationships with members of the covert network and leveraging those relationships to gain the desired effect on the network. A review of the literature highlights a considerable gap in research on the use of node insertion within the context of covert networks. This gap in literature serves as the motivation for this study as situations might dictate the need for an agent or sensor to be inserted into a covert network for the purpose of future disruption or further intelligence gathering.

1.2 Problem Statement

With the current lack of research in the area of node insertion with respect to covert networks, there exists a need for effective and efficient methods to determine the most advantageous node insertion scenarios for a given covert network. Within the context of this research, a node insertion scenario represents one, two, or three members of the network that an agent who is inserted into the network should target for relationship development and information gathering.

The motivation for this research is the need to evaluate the extensive number of insertion scenarios that result from large covert networks with over 150 members. For instance, a network consisting of 200 members results in 200 scenarios targeting one individual (one-target scenarios), 19,900 scenarios targeting two individuals (two-target scenarios), and 1,313,400 scenarios targeting three individuals (three-target scenarios). This results in 1,333,500 different scenarios to analyze to determine which scenarios are the most advantageous to pursue, and this analysis requires many hours of computational time. As the network size grows and the number of possible targets per scenario increases, the number of possible insertion scenarios grow according to a power law. This in turn greatly increases the computational time required to evaluate all possible insertion scenarios.

1.3 Research Objectives

The main objective of this research is to provide a method to reduce the computational time required to produce a list of the most advantageous insertion scenarios within a covert network, while maintaining an adequate overall level of accuracy. To accomplish this objective, this study provides a screening heuristic that reduces the solution space by screening out unfavorable scenarios and then efficiently searches the reduced space for the most advantageous scenarios. Three screening heuristics, differing in their respective screening criteria, are presented in this paper: neighbor-access screening, utility-score screening, and measure screening.

To accomplish the main objective, the study proposes a utility function to score each scenario and serves as the objective function that the heuristic seeks to maximize. The utility function consists of two subfunctions that quantify the benefit and risk

associated with each insertion scenario. A weighted combination of common SNA measures is used to calculate the benefit and risk scores for a given scenario, and weights can be tuned depending on the operational context. The overall utility score of the scenario is the weighted difference between the benefit and risk scores.

The goal of this research is to provide intelligence and law enforcement agencies with a method to quickly develop a list of the most advantageous node insertion scenarios within a large covert network. The outputs of the screening heuristics are intended to be an initial step in the overall evaluation process of potential insertion scenarios and are not intended to replace decision making without further analysis of the scenarios. The list of advantageous scenarios produced by the screening heuristic will require additional analysis by intelligence and infiltration subject matter experts (SMEs). Using the proposed screening heuristics will reduce the number of scenarios requiring this additional analysis, saving time.

1.4 Research Scope

The focus of this study is to develop a screening heuristic to efficiently and accurately develop a list of advantageous insertion scenarios, given a specified covert network. Insertion scenarios consist of one to three individuals within the network, with whom an inserted agent should seek to develop a relationship in order gain a desired effect. This research does not evaluate the network's response to the agent's insertion or take into consideration any security measures of the covert network to seek out infiltrators. The analysis of the actual process of and techniques used to insert an agent into a covert network are outside the scope of this study. Classified information, to

include current classified techniques to infiltrate a covert network are not used for this research.

1.5 Assumptions and Limitations

The assumptions used for this methodology include the following:

1. Attributes of the network's nodes and edges are not considered. This includes geographical information of nodes. The edges within the network are undirected and unweighted.
2. Networks are known with 100% certainty. All members and their respective connections are known and present within the network at the time of analysis. This is counter to the real world, where information on covert networks contains a level of uncertainty and existing members of a network will be unknown to the analyst.
3. The networks used for this research are not modeled dynamically, and therefore represent the network at a given point in time.
4. Due to the lack of access to large covert network datasets, theoretical graph generators will be used to generate the necessary networks for testing and analysis.
5. Testing of the screening heuristic will be limited to networks consisting of 500 nodes or less. This is due to both time and access to computers with required processing power.

1.6 Thesis Organization

This thesis contains four additional chapters. Chapter II provides an overview of the supporting literature used during this research. Key topics covered in Chapter II include an overview of the key SNA measures, screening heuristics, and an overview of current research applying SNA techniques to covert networks. Chapter III develops the methodology for the research and discusses the development of both the utility function and screening heuristic. The methodology is then applied to the September 11th Terrorist Network case study. Chapter IV discusses the testing procedure, results, and analysis of the screening heuristic's performance. Chapter V provides a discussion of the conclusions gained from the research and provides areas for future research.

II. Literature Review

2.1 Chapter Overview

This chapter provides an overview of current literature on the topics of covert networks and SNA. The chapter starts with an overview of characteristics of covert networks and is followed by discussions of key SNA measures used to evaluate networks at both the node and network level. The chapter concludes with a review of current research applying SNA techniques to the destabilization of covert networks.

2.2 Covert Networks

According to The Mitchell Centre for Social Network Analysis covert networks "are social networks, which have one or many elements of secrecy about it" [6]. Covert networks form around a given activity, ideology, or purpose that is either illegal, dangerous, or otherwise deemed unacceptable by societal norms. Examples of covert networks include protest movements, underground resistance movements, espionage groups, people that participate in certain sexual behaviors, extremist, terrorists, and criminal organizations [6].

For this research the term "covert networks" refers to terrorist, extremist, and criminal organizations, with the primary focus on terrorist organizations. This is a common use and scope of the term throughout the literature reviewed.

2.2.1 Covert Network Structure

The consensus amongst researchers is that most terrorist organizations take on a hub-and-spoke or cellular structure [7]. Initially, this was not the case as the structures of these organizations were more hierarchical in nature. Yet, mainly through the advent of

the Internet, these organizations gained a global reach resulting in distributed and decentralized operations forcing the organizations to adopt the hub-and-spoke structure commonly seen today [8].

The hub-and-spoke or cellular structure of covert networks serves as a protection mechanism for the organization, limiting the effects of disruption and helping to maintain secrecy. Communication and knowledge of operations between cells are limited, with only the cell leadership having operational connections outside of the cell. Cell members only maintain limited intra-cell connections [7]. A few cell members do maintain what Tsvetovat et al. [9] refer to as "sleeper links." These links are non-operational connections to members of other cells, usually in the form of family ties or shared experiences like training, that can be activated as the need arise. These "sleeper links" and other network characteristics help to maintain redundancies so that flow of information or resources is not disrupted if a critical node or cell is discovered and removed from the network [9], [7].

Network hubs, the more connected members of the network, coordinate operations and facilitate information flow between cells as needed. The hubs are usually not leaders within the network but help to reduce the distance that information is required to travel between leadership and cells. Removing hubs from the network can cause major disruptions to the network [7], but Tsvetovat et al. [9] argue that activation of the "sleeper links" can help to mitigate the disruption.

Despite terrorist organizations sharing an overarching hub-and-spoke structure, individual organizational structures do differ in many ways. This is highlighted by Carley [10] in her comparison of the structures of al Qaida and Hamas. Both

organizations have an underlying cellular structure, but the organization and make up these cells differ between groups. Table 1 highlights the key differences between these groups.

Table 1. General Features of Terrorist Groups [10]

Feature	al-Qa'ida	Hamas
Organizational structure	Top-hierarchy with underlying cellular for central al-Qa'ida connected hierarchically to related groups like JI	Matrix – by region and function
Cells	Distributed skills in cells	Same skills in cells
Missions - small	Planned, executed, funded by cell	Planned by function, executed locally, funded regionally
Missions - large	Planned, funded, and executed by multiple cells under temporary hierarchical coordination structure	Matrix planning and funding coordinating execution by multiple cells
Cell orientation	Mission – unlimited number of missions	Function – there are five basic functions
Cell leaders	More experience/ broader knowledge	More experience/ similar knowledge / political connections
Connection among cell leaders	Similarity (historical)	By function and region
Locus of control	Distributed - cells can operate independently for small missions	Predominantly centralized - cells coordinated by regional and functional leader
Directives	Can come from higher up	Always come from higher up
Executive group	Broad	Narrow
Members of executive group (power in terrorist group)	Some cell leaders	Most functional and regional leaders
Political power in community	Varies	Regional leaders are often powerful
Integration with community	Moderate to low Hidden	Reasonably high Infrastructure cells legitimate businesses Operational cells infiltrated into police, military etc.
Connection to other terrorist groups	Hierarchical	Lateral – through intermediaries

The al Qaida organization maintained an overarching hierarchy built around Usama bin Laden. Below bin Laden was a council of his deputies that oversaw specific functions in the areas of religion, military, finance, and media. Below bin Laden and his deputies were the dispersed and decentralized operational cells of the organization. These cells were multi-functional in the sense that members of the cells contained the various skill sets required for the cell to be operationally effective, independent of the rest of the organization. These cells were free to plan, fund, and conduct small operations autonomously. Larger operations, like the September 11th Attack, were conducted by

multiple cells through the coordination and funding provided by bin Laden and his deputies [10].

Hamas's cellular structure is organized based on region and function. Each region (West Bank, Egypt, etc.) contains functional cells or groups: internal security group, uprisings group, suicide bomber group, professional killer group and a support group [10]. The functional groups report to both a regional leader and an overarching functional leader at the Hamas headquarters. The regional and overarching functional leaders report directly to the Hamas leader. For operations, coordination between groups within a region is required to ensure the required skillsets are available. In certain regions, the cells may remain dormant until activated for an operation [10].

The United States military maintains a similar view on the structures of covert networks (referred to in doctrine as "threat networks") as academia. In *Joint Publication 3-25: Countering Threat Networks* (JP 3-25), the compartmentalization of the network is critical to the network's survival. The goal of this compartmentalization is to reduce any single individual's knowledge on the network's operations or members. This is to reduce risk to the network if an individual is detained [11]. Figure 1 depicts a notional network structure as presented in JP 3-25.

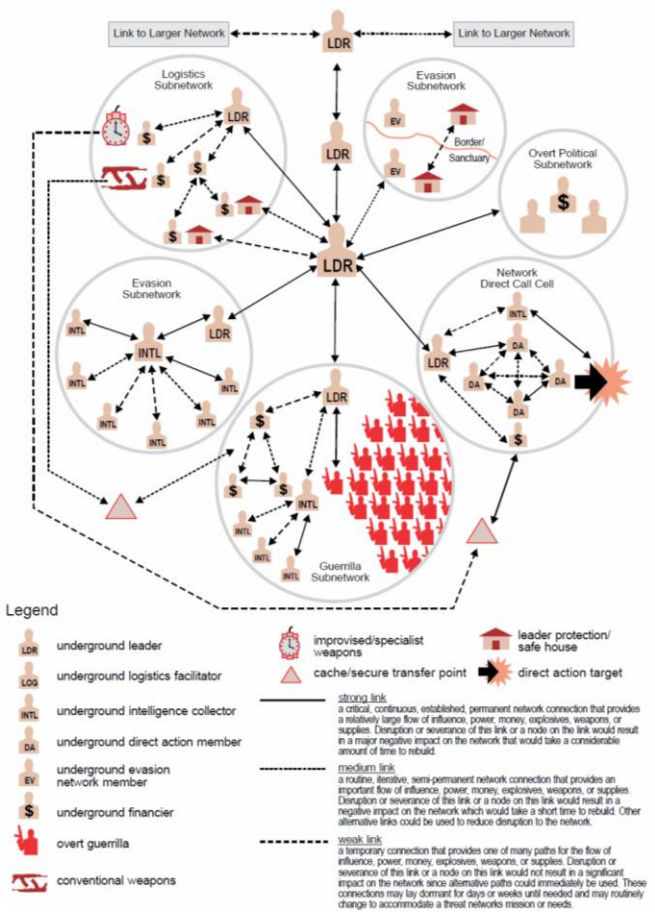

Figure 1. Depiction of a Notional Covert Network from Joint Publication 3-25 [11]

2.3 Social Network Analysis (SNA)

SNA is "the analysis of systems of social relationships represented by networks" [12]. SNA is rooted in graph theory, which models graphs or networks as a system of points and lines. Information about a network is gained through graph theory's leveraging of information contained within the network's structure and properties [12].

In SNA, social networks and organizations are represented as a system of points, called nodes, and lines, called edges. Members of a social network are represented by nodes and the connections between the members are represented by the edges. The information about the network is normally contained within a square adjacency matrix,

where the rows and columns represent the individuals, and values in the matrix cells represent a connection [12]. Within the network, nodes can also represent collectives such as tribes, countries, or teams. Characteristics about nodes are captured in the form of attributes and help to distinguish between the nodes [13]. By weighting edges, the strength of a relationship can be captured. Directional edges can be used to represent information flow within the network [12].

Borgatti et al. [13] describes the three levels of social network analysis: dyad, node, and network. At the dyad level of analysis, pairwise relationships between network members are the focus. Node level analysis aggregates dyad measures to determine insights into the individual member, such as the member's positioning or influence within the network. The final level of analysis is the network level. At this level, measures seek to describe the network as a whole, such as the density of relationships or the diameter of the network [13]. The following sections describe key measures at the network and node level, followed by a discussion of various network types.

2.3.1 Network Level Measures

The initial set of network measures consists of *density*, *diameter*, and *average path length*. The density of a network is the percentage of possible connections present within a network. If a network contains five nodes there are ten possible connections ($n(n-1)/2$) within the network. If there are five edges present, this results in a density of 0.5 [13]. The diameter of the network is greatest path distance between any two nodes in the network. The average path length is similar to diameter but calculates the average of all shortest paths between all the nodes in the network [14].

The next network measure, the *clustering coefficient*, seeks to determine the level of clustering within a network. Jackson [14] describes clustering as "the extent to which my friends are friends with one another." The clustering coefficient was introduced by Watts and Strogatz [15] in 1998 and is based on the transitivity of a set of three nodes. A set of three nodes are considered transitive when all possible edges between the three nodes are present. The clustering coefficient looks at all cases where two edges (*ij* and *ik*) originate from the same node *i*, and determines how often, on average, the edge *jk* exists. The clustering coefficient is also calculated for individual nodes in a similar manner. Clustering coefficients help in identifying communities within a network [13], [14], [15].

Modularity is a similar measure to the clustering coefficient, as it seeks to determine the level of communities within a network. Newman and Garvin define modularity as measuring "the fraction of the edges in the network that connect vertices of the same type (i.e., within-community edges) minus the expected value of the same quantity in a network with the same community divisions but random connections between the vertices" [16]. The modularity value ranges from -0.5 to 1. If the number of within-community edges is less than the random edges, modularity will be negative. If modularity is close to one, the network has a strong community structure. Networks with a modularity greater than 0.6 have a clear community structure [16], [17], [18].

The final network level measure is *degree assortativity*, which measures the correlation between the degree of a node and the average degree of its neighbors. If this correlation is positive and the assortativity coefficient is close to one, the network is assortative, meaning nodes of high degree tend to be connected to other nodes of high

degree. A disassortative network has a negative correlation, meaning nodes of high degree are connected to many nodes of low degree. Assortativity can also be calculated for various node attributes, which is useful in the study of homophily [18], [19].

2.3.2 Node Level Measures

Borgatti et al. [13] define *centrality* as "a class of theoretical constructs that characterize a node's position within the network structure," and centrality analysis as "the process of scoring each node in the network according to its structural importance." The driving idea behind centrality measures is that key members of a network are centrally located within the network, and centrality measures allow analyst to identify these key members. Jackson [14] notes that there are four categories of centrality measures, based on their underlying statistical theory: degree, closeness, betweenness, and neighbors' characteristics.

The "simplest and perhaps the most intuitively obvious" form of centrality is *degree centrality*, which is based on the node's degree, or the number of edges that connect with the node [20]. The underlying idea is that the more connections an individual has, the more central they are positioned in the network, and the more individuals they can influence compared to an individual with a low degree. Degree centrality is commonly used with undirected networks where any arc weights are ignored. Degree centrality can also be used with directed networks in the form of in-degree and out-degree centrality. It is calculated by summing the row values within the adjacency matrix [21]. Degree centrality is commonly normalized by dividing by $(n-1)$ [14].

Jackson highlights a weakness of degree centrality by emphasizing that this form of centrality does not take into consideration the quality of the adjacent nodes. For

example, an individual with a lower degree but whose links are with influential individuals are more central in the network than an individual that has a higher degree but whose adjacent nodes are uninfluential and positioned on the periphery of the network [14].

The next category of centrality measures is *closeness centrality*. As the name implies, closeness centrality captures how close a node is to all other nodes in the network. When discussing information flow across a network, individuals with a higher closeness centrality will receive information quicker as it spreads across the network [21]. The equation for the normalized closeness centrality is

$$c_i^c = \frac{(n-1)}{\sum_{j \in V: j \neq i} d(i,j)} \tag{2.1}$$

where *d(i, j)* is the shortest path distance from node *i* to node *j* [20].

Jackson identifies a second form of closeness centrality called decay centrality,

$$c_i^d = \sum_{j \in V: j \neq i} \delta^{d(i,j)} \tag{2.2}$$

that utilizes the decay parameter, δ, with a value of $0< \delta<1$. As δ approaches zero, more weight is given to closer nodes [14].

Betweenness centrality is the third category of centrality as described by Jackson. Freeman [20] introduced the concept of betweenness centrality in 1977 and describes it as a measure of how well an individual is positioned between others in the network allowing them to "facilitate, impede or bias the transmission of messages." The equation for the normalized form of betweenness centrality is

$$c_i^b = \sum\nolimits_{k \neq j; i \notin \{k,j\}} \frac{P_i(kj)/P(kj)}{(n-1)(n-2)/2} \tag{2.3}$$

where $P_i(kj)$ is the number of shortest paths between *k* and *j* that *i* falls, $P(kj)$ is the total number of shortest paths between *k* and *j*, and *n* is the total number of nodes. As a node's betweenness approaches one, the more central the node is considered [14].

Stephenson and Zelen [22] argued that betweenness centrality is narrow in its scope as information and communication does not travel only on shortest paths within a social network. They proposed a calculation that determines the centrality of a node based on the total number of paths on which a given node is located. They refer to this measure as information centrality [22]. When working with an unweighted graph the $n \times n$ matrix $\boldsymbol{B} = (b_{ij})$ is derived from the following:

$$b_{ij} = \begin{cases} 0, & \text{nodes } i \text{ and } j \text{ are incident} \\ 1, & \text{otherwise} \end{cases} \tag{2.4}$$

$$b_{ii} = 1 + \text{degree of node } i \tag{2.5}$$

The information centrality of node *i* (c_i^I) is

$$c_i^I = \left(\frac{1}{k_{ii} + (T - 2R)/n}\right) \tag{2.6}$$

where $\boldsymbol{K} = (k_{ij}) = \boldsymbol{B}^{-1}$, $T = \sum_{j=1}^{n} k_{jj}$, and $R = \sum_{j=1}^{n} k_{ij}$ [22].

The final category of centrality, as discussed by Jackson, consists of those measures that include the characteristics of a node's neighbors within the calculation. These measures are based on the idea that one's importance or influence within a network is not merely based on the quantity of connections or position within the network, but the quality of those connections [14].

Bonacich describes *eigenvector centrality* as "a measure of centrality in which a unit's centrality is its summed connections to others, weighted by their centralities" [23].

Borgatti describes eigenvector centrality as a measure of popularity: nodes with high eigenvector centrality are connected to other nodes that are well connected. Borgatti continues to describe how eigenvector centrality can be used to measure risk within a network, using the risk of disease transmission within a network as an example [13]. The equation for eigenvector centrality (c_i^e) in matrix notation is

$$\boldsymbol{A}\vec{x} = \lambda\vec{x} \tag{2.7}$$

where $\boldsymbol{A}$ is the adjacency matrix of the graph, λ is the largest eigenvalue, and $\vec{x}$ is the vector of eigenvector centralities. c_i^e is the i^{th} element of $\vec{x}$ [23].

A similar centrality to eigenvector centrality was proposed by Leo Katz in 1953 and used to calculate a node's status or influence within a network. This has become known as *Katz centrality*. This centrality measures the number of nodes that are immediately connected to the node of interest. It then measures the number of nodes that connect to the node of interest through the immediate neighbors. These distant nodes are penalized by α^k, where k is the distance from the distant node to the node of interest, and α is an attenuation factor that is set less than $1/\lambda_{max}$ [24], [25].

In 2013 Qi et al. presented the *Laplacian centrality* measure that utilizes Laplacian energy to measure the "centrality of a vertex based on the idea that the importance of a vertex is related to the ability of the network to respond to the deactivation or removal of that vertex from the network" [26]. This centrality measure looks to capture the amount of Laplacian energy that the network loses when a node and its corresponding edges are removed from the network. The greater the energy lost after node removal, the greater the importance of the node in the network [26].

Laplacian centrality is calculated using the Laplacian matrix of the network. The Laplacian matrix of network *G* is calculated by $\boldsymbol{L}(G) = \boldsymbol{D}(G) - \boldsymbol{A}(G)$, where $\boldsymbol{D}(G)$ is a diagonal matrix where d_{ii} is the degree of node *i* and $\boldsymbol{A}(G)$ is the adjacency matrix of network *G*. Utilizing the eigenvalues (λ_i) of the Laplacian matrix, the Laplacian energy for *G* ($E_L(G)$) is calculated by

$$E_L(G) = \sum_{i=1}^{n} \lambda_i^2 \tag{2.8}$$

Node degree can also be used to calculate $E(G)$:

$$E_L(G) = \sum_{i=1}^{n} (d_i^2 + d_i) \tag{2.9}$$

where d_i is the degree of node *i*. This results in the Laplacian centrality of node *i* as

$$c_i^L = (\Delta E)_i = E_L(G) - E_L(H) \tag{2.10}$$

where *H* is the resulting network after node *i* is removed from network *G* [26]. The above equation can be further defined as

$$c_i^L = (\Delta E)_i = d_G^2(i) + d_G(i) + 2 \sum_{j \in N(i)} d_G(j) \tag{2.11}$$

where *N(i)* is the set of neighbors of node *i* [26].

There are many more measures within SNA to determine the importance or status of nodes, including variations on the above. Guzman et al. [27] researched the relationships between 24 common measures to determine if there is any statistical dependence between the measures. Table 2 displays the four groups found by the study where measures in each group are highly correlated with each other [27].

Table 2. Groups of Highly Correlated Measures [27]

Group 1	Group 2
Clustering coefficient	Betweenness centrality
Soffer's clustering coefficient	Stress centrality
Squares clustering coefficient	Length-scaled betweenness
	Linearly-scaled betweenness
	k -betweenness
	Random walk betweenness
	Proximal source betweenness
Group 3	**Group 4**
Load centrality	Degree Centrality
Proximal target betweenness	Pagerank

Table 3 shows the measures that were found to be uncorrelated with any of the other 24 measures.

Table 3. Uncorrelated Network Measures [27]

Measure
Closeness centrality
Eigenvector centrality
Communicability centrality
Simple diversity
General diversity
Communicabilitybetweenness
Current flow betweenness
Approx. current flow betweenness
Closeness vitality
Average neighbor degree

Borghatti *et al.* [28] researched the robustness of centrality measures in the presence of random error in network data. The study used node addition, node deletion, edge addition, and edge deletion to replicate the error within the network, and their effects on degree, betweenness, closeness, and eigenvector centralities. The results of the study show that accuracy of the measures declines "smoothly and predictably with the amount of error." The authors suggest that this relationship between accuracy and error will allow for the construction of confidence intervals for centrality measures. Another

key insight from the study is that measures of large dense networks are more robust for all error types except edge deletion [28]. This research is important to using SNA measures on covert networks, as their secretive nature makes it difficult to collect all information on the network.

2.3.3 Network Structures

A small-world network is characterized by a high clustering coefficient, similar to a lattice graph, and a small average path length between nodes similar to a random graph. The common "six degrees of separation" concept is based on a small-world network [15]. Watts and Strogatz [15] proposed a method of generating small-world networks. The model initializes with a lattice graph of n nodes that are connected to their k nearest neighbors. Then moving around the ring clockwise, each edge is selected. With a probability of p, the edge is rewired to another node that is uniformly selected. With a probability of $(1-p)$ the edge is not rewired. Figure 2 shows the random rewiring process where n=20 and k=4 [15].

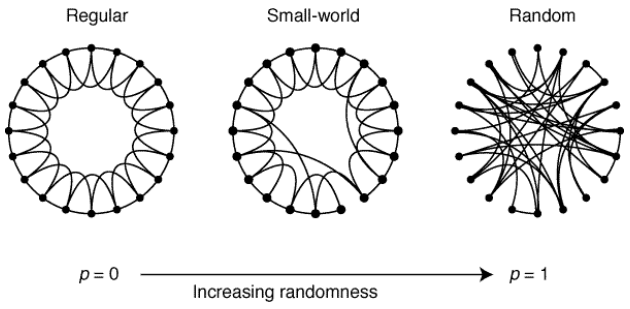

Figure 2. Random Rewiring Procedure from a Regular Ring Lattice to a Random Network *[15]*

Random graphs are graphs where edges between nodes are added to a network based on some probability. One of the most famous models for generating a random

graph was introduced by Paul Erdős and Alfred Rényi in 1959. Their model uses two parameters, *n* for the number of desired nodes in the network and *p* the probability that an edge is formed. The model initializes by creating the *n* nodes. Then the model iterates across every possible edge and adds it to the network with a probability of *p*. Edge insertion is independent of the other edges in or out of the network [30]. Random graphs might seem simple, but they help to provide insight into properties of social networks [14].

The third network structure is the scale-free network. These networks have degree distributions that follow a power law distribution. Due to the nature of this distribution and the size of its tails, highly connected hub-nodes have a high probability of occurring [29]. A modeling method for scale-free networks was presented by Barabási and Albert. Unlike Erdős–Rényi and Watts-Strogatz network models, the Barabási–Albert model does not start with all *n* nodes present. The model starts with *m* nodes and at each iteration a new node is added with *m* edges that connect to nodes already present in the network via preferential attachment. The probability, *P*, that a new node will connect to node *i* depends on the connectivity (k_i) of node *i*, which results in $P(k_i) = k_i \Big/ \sum_j k_j$ [29].

The final network structures are the connected caveman graph and one its variations, the relaxed caveman graph. The caveman graphs are used to study clustering. The connected caveman graph originates with *l* cliques consisting of *k* nodes. One edge is rewired from each clique to connect all the cliques into a loop as depicted in Figure 3.

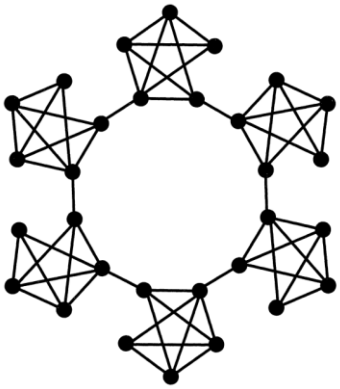

Figure 3. Connected Caveman Graph [31]

The relaxed caveman graph initializes with the same *l* cliques and *k* nodes but incorporates a probability *p* that determines if each edge within the clique is rewired to outside the clique [32]. This network structure closely resembles the cellular network structure commonly seen in covert networks.

2.4 Current Research of Covert Networks

Since the terrorist attacks on September 11, 2001, there has been an increased interest into the research of covert networks, specifically terrorist networks, and methods to counter them. After the attacks, government intelligence and defense agencies lacked the skillset and abilities to properly analyze these networks. They, along with academia, turned to SNA to fill this capability gap [27]. The following section discusses the challenges that researchers face when applying SNA to covert networks along with a review of current research focused on network disruption and intelligence gathering.

2.4.1 Data Challenges

In her review of current literature concerning terrorist networks, van der Hulst [7] discusses the challenges facing current researchers when trying to utilize SNA principles to study terrorist networks, and by extension covert networks. She identifies two types of

researchers that are interested in the terrorist network problem set: operational analysts and scientific researchers. The operational analysts are those individuals working for government intelligence agencies that have access to classified information on terrorist networks, but due to the classification, any research results they produce cannot be made publicly available. The second group are members of academia who have the expertise to model network behaviors but lack access to classified information and datasets, forcing them to rely on open-source data or media reports. These data sources are usually incomplete or wrong [7].

Diviak [33] expands on the above issues affecting the research of covert networks. Due to inherent nature of covert networks and their members emphasis on secrecy and compartmentalization, collecting complete datasets on these networks is nearly impossible. Most unclassified datasets available to researchers are developed after an attack has occurred, usually through the consolidation of information provided in news reports and unclassified government reports. In many cases, these data sources are incomplete or conflicting [33].

van der Hulst [7] discusses the effects of the above data challenges on the overall covert network research effort: "In the absence of actual network metrics, however, most of the work remains limited to the discussion of social networks as a paradigm, some theoretical arguments, or basic qualitative analysis." She continues by saying that many papers are "essentially descriptive (but limited) or outline the potential value of SNA to study clandestine networks," resulting in few empirical studies or studies that test a proposed hypothesis [7]. Asal and Rethemeyer [34] describe the research field as being mainly "theory building with little empirical verification."

2.4.2 Network Disruption through Node Deletion

Current research on the topic of disruption of covert networks centers around the removal of key nodes within the network. Carley et al. [35] use SNA and multi-agent models to research how to destabilize covert networks. The study highlights the difficulty of destabilizing large and distributed networks as well as those where the network members are connected on various dimensions. In the study the authors model networks dynamically and use meta-matrix representation. Both techniques enable the model to capture a network's ability to adapt, learn, and change over time [35].

Carley et al. [36] continues this line of research through the development of DyNet, a tool that is used to simulate covert networks dynamically and uses a meta-matrix modeling technique. Using DyNet, the study was able to simulate the effects of various targeting and disruption strategies. The outcome of the research revealed that if the individual removed from the network has a high cognitive load or task exclusivity this has a greater effect than removing someone who is considered central to the social network. Both removal strategies are slightly better than random node removal. In the optimal organization structure where tasks and cognitive load are evenly dispersed throughout the network, random removal increases in effectiveness.

Tsvetovat et al. [9] also use multi-agent simulations to test network disruption strategies. Their findings showed similar results to the above study, revealing that removing experts and other key members based on role creates more permanent damage to a cellular-structured network then the isolation of well-connected individuals. The study also revealed that a cellular network has an emergent healing behavior that can negate the effects of node loss after some time [9].

Geffre et al. [37] developed a method to determine a quantitative value of criticality based on three measures: "(1) the members' social connectedness across multi-layered affiliations, (2) their involvement in operations, and (3) their emergence during periods and at locations of interest." This multi-layered approach is displayed in Figure 4.

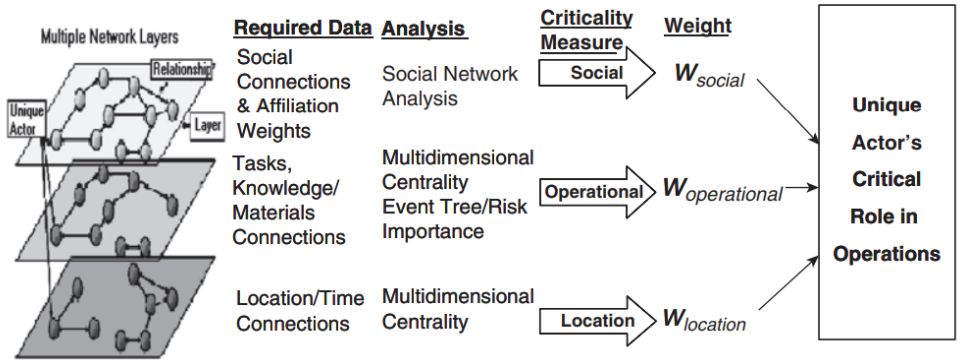

Figure 4. Multi-Layered Approach used by Geffre et al. *[37]* to Determine an Actor's Critical Value

2.4.3 Network Disruption through Node Insertion

There currently is a lack of research into utilizing node insertion as a disruption technique. One study conducted by Johnstone [38] utilizes a risk-based approach to determine locations within a covert network to insert an agent for the purpose of intelligence gathering or future disruption. In the study, Johnstone calculates a utility score for each possible insertion scenario, where a scenario consists of one to four members of the network of interest that are "targets" with whom an inserted agent should develop a relationship. The utility consists of weighted benefit and risk values. Each of these values are calculated using degree, subject matter expert (SME) rating, closeness centrality, and Laplacian centrality. The method shows promise for identifying potential

node insertion scenarios but becomes computationally expensive when utilized on large networks [38].

2.5 Chapter Summary

This chapter provided an overview of the current literature with respect to covert networks and SNA. The information gained through the literature review process will be applied in the next chapter with the development of the study's methodology.

III. Methodology

3.1 Chapter Overview

This chapter details the research methodology and begins with a discussion of the notation used throughout this chapter. Second, the method for insertion scenario scoring is explained. This scoring method uses equations consisting of various SNA measures to calculate risk and benefit scores. These scores are combined to create an overall utility score for each insertion scenario. Next, the three screening heuristics are discussed to include the screening criteria used by each heuristic to reduce the solution space. This is followed by the presentation of the search portion of the heuristics that operate on the reduced solution. Finally, the methodology is applied to case study using the September 11th Terrorist Network.

3.2 Notation

This paper uses the following terminology and notation throughout. Social networks are referred to both as *networks* and *graphs*. Members of the social network will be referred to as *nodes, targets, members,* or *individuals*. *Edges, connections,* and *relationships* refer to the social links between members of the social network. The node that is inserted into the network is either the *agent* or the *sensor*.

Each social network or graph of interest is defined as $G(V, E)$, where V is the set of all nodes and E is the set of all edges. Nodes are represented as v_i, where i is the index of the node. Edges are represented as $e_{i,j}$, where i and j are the indices of the nodes that the edge connects. The inserted node, or the agent, is represented as v_a. The number of nodes in graph G is $|V| = n$ resulting in $V=\{v_1, v_2, \ldots, v_n\}$ and $E=\{e_0, e_{0,1}, \ldots, e_{i,j}\}$.

To represent a generic node insertion scenario, x_s is the notation used, where s is the set of all targets within the scenario. A more explicit representation of the insertion scenarios is also used where x_i is used for a one-target scenario, $x_{i,j}$ for a two-target scenario, and $x_{i,j,k}$ for a three-target scenario, where i, j, and k are the indices of the nodes with whom the agent targets for relationship development (network edge formation). When the agent is inserted into the network, the network's definition is modified to be $G^*_{x_s}(V^*_{x_s}, E^*_{x_s})$.

Throughout this chapter, a trivial social network is used to help explain the different aspects of the methodology (Figure 5). This network consists of seven individuals and eleven edges, resulting in a network density of 0.52.

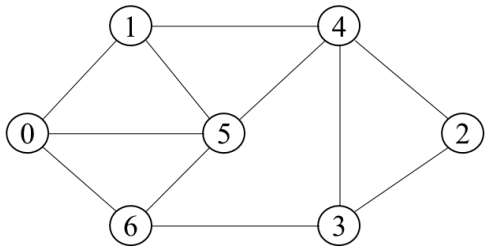

Figure 5. Example Network

3.3 Utility Function

The purpose of the utility function is to provide a measure of both benefit and risk, as well as an overall score for each insertion scenario. Though not the primary focus of this research, the utility function plays an important role, as it serves as the objective function for the heuristic. The utility function utilized in this research is motivated, with exceptions, by the similar utility function used by Johnstone [38] and is discussed in Chapter II.

The utility function developed for this thesis consists of a weighted benefit score and a weighted risk score defined as

$$U_{x_s} = w_B B_{x_s} - w_R R_{x_s} \tag{3.1}$$

where U_{x_s}, B_{x_s}, and R_{x_s} are the utility, benefit, and risk scores, respectively, associated with node insertion scenario x_s. The weights w_B and w_R are associated with benefit and risk, respectively, and sum to one. Both benefit and risk are calculated using various SNA measures and are discussed in greater detail below.

3.3.1 Benefit Score

For this research, we assume that the purpose of any node insertion is for intelligence gathering on the network. Due to this assumption, two common SNA measures are utilized to calculate the benefit score for each node insertion scenario: eigenvector centrality and betweenness centrality. As discussed in Chapter II, eigenvector centrality determines the centrality of individuals within a social network, not by the mere numbers of connections an individual has, but by the quality of those connections. An individual with a high eigenvector centrality is central to the network because they are connected to other important individuals within the network. The underlying principle of betweenness centrality is that individuals that are positioned on a higher number of shortest paths between network members have a greater access to information flowing through the network than those individuals positioned on fewer shortest paths. Since the assumed goal is to collect information on the network, the inserted agent needs to develop relationships with individuals who have access to important members of the network and are better positioned to intercept information flow between members.

The formula for the benefit score is constructed as

$$B_{x_s} = w_e \sum_{i \in S} c_i^e + w_b \sum_{i \in S} c_i^b \tag{3.2}$$

where *S* is the set of targets in scenario x_s, c_i^e is the eigenvector centrality for target *i*, and c_i^b is the betweenness centrality of target i. These centrality measures are calculated using $G_{x_s}^*(V_{x_s}^*, E_{x_s}^*)$. w_e and w_b, where $w_b = 1 - w_e$, are the respective weights for the two centralities. The weights allow the analyst to emphasize one measure over the other within the benefit score. Instances may arise where the agent's access to important individuals within the network is more import than positioning to intercept information flow. In this case, w_e can be set higher than w_b.

3.3.2 Risk Score

The risk calculation uses two SNA measures and a cost factor for each relationship the agent develops. The first measure used to determine risk of the insertion scenario is closeness centrality (c_a^c). As discussed in Chapter II, closeness centrality measures how close a member of the network is to all other members of the network by utilizing the shortest path distance from the node of interest to all other nodes in the network. A member is more central within a network if the individual has a higher closeness centrality value. For node insertion, executing an insertion scenario that results in the agent having a high closeness centrality could increase the likelihood of the agent being detected.

The second SNA measure used to calculate risk is degree assortativity, a network level measure. As discussed in Chapter II, degree assortativity (γ) is a measure of the distribution of the degrees of connected nodes. Networks with a positive assortativity contain a large portion of high degree nodes connected to other nodes of high degree.

Within the context of node insertion, if the insertion scenario does not conform to the existing connection patterns within the network, there will be a larger change in degree assortativity, and the agent could be more susceptible to detection. By using the change in degree assortativity ($\Delta\gamma$) from before and after agent insertion within the risk calculation, the change in the network's status quo is captured.

The final aspect of the risk calculation is a standard cost factor (δ), which represents the costs associated with establishing each relationship in the insertion scenario. The value of the cost factor is established by the analyst in coordination with intelligence SMEs and ranges between zero and one. The cost factor plays an important role within the utility function, where it is multiplied by the number of relationships the agent is to establish in the insertion scenario. Without the cost factor, the utility function will not capture the risk associated with developing multiple relationships within the covert network. As the number of relationships the agent pursues increases, the likelihood of the agent being detected is expected to increase. The cost factor tries to capture this risk.

The equation for the risk calculation is

$$R_x = w_c c_a^c + w_\gamma(|\Delta\gamma|) + r_{x_s}\delta \tag{3.4}$$

where c_a^c is the closeness centrality of the agent after insertion and r_{x_s} is the number of targeted relationships in the insertion scenario x_s ($r_{x_s} = |S|$). The weights for closeness centrality and change in degree assortativity are w_c and w_γ, respectively, and sum to one.

3.3.3 Utility Function Applied to Example Network

For the network example in Figure 5, there are sixty-three possible insertion scenarios when allowing the agent to create at most three relationships. There are seven

scenarios where the agent develops a single relationship, twenty-one scenarios for two relationships, and thirty-five scenarios for three relationships. For this example, only thirteen of the possible sixty-three scenarios are analyzed. These thirteen scenarios were selected to ensure certain points are made during the following explanations.

To calculate the utility score for each scenario, the initial step is calculating the degree assortativity for $G(V, E)$, and is only required to be calculated once per network, as this value will be the same across all possible scenarios. For the example network, the initial degree assortativity is -0.196.

The remaining calculations within the utility function require $G_{x_s}^*(V_{x_s}^*, E_{x_s}^*)$, the updated network once the agent is inserted and relationships are created between the agent and the scenario's targets. The next step is to calculate the eigenvector and betweenness centrality for each target in the scenario, the closeness centrality for the agent, and the new degree assortativity. Once the desired measures are known, the scenario's benefit, risk, and utility scores can be calculated. Once a scenario's scores are calculated, the agent and its respective relationships are removed from the network, resetting back to $G(V, E)$.

Calculating the eigenvector, betweenness, and closeness centralities for every possible $G_{x_s}^*(V_{x_s}^*, E_{x_s}^*)$ becomes computationally expensive, especially for large, dense networks. Yet, capturing the effect of the agent's insertion on the network and targets could provide valuable information that can be utilized by the intelligence analysts in post-analysis. For example, a major shift in the targets' eigenvector centrality could highlight a possible scenario where the new relationships and the agent could be easily

detected. This is outside the scope of this research but can serve as a topic for future research.

Figure 6 shows the first insertion scenario, x_0, where the agent, v_a, develops a relationship with the target, v_0. The dotted line represents the new relationship that the agent will pursue in the given scenario.

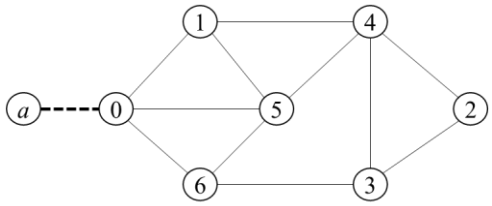

Figure 6. Network Representation of Scenario x_0

Once the agent is inserted into the network and the corresponding relationships are developed, the centrality measures and change in degree assortativity of the network are calculated. The new degree assortativity for $G_{x_0}^*(V_{x_0}^*, E_{x_0}^*)$ is -0.304, resulting in $\Delta\gamma = 0.109$. The following equations show the calculations for associated risk, benefit, and utility scores for scenario x_0 when using equal weights of 0.5 and a cost factor of 0.3.

$$B_{x_0} = 0.5(0.393) + 0.5(0.111) = 0.252 \quad (3.5)$$

$$R_{x_0} = 0.5(0.412) + 0.5(0.109) + 1(0.3) = 0.561 \quad (3.6)$$

$$U_{x_0} = 0.5(0.252) - 0.5(0.561) = -0.155 \quad (3.7)$$

Table 4 shows the benefit, risk, and utility scores for the selected insertion scenarios for the example network. The top three scenarios based on U_{x_s}, in order, are $x_{2,3}$, $x_{2,3,4}$, and x_3.

Table 4. Example Network Scenario Scores, with Top 3 Scenarios Highlighted

Scenario	$\lvert\Delta\boldsymbol{\gamma}\rvert$	Benefit Score B_{x_s}	Risk Score R_{x_s}	Utility Score U_{x_s}	Ranking (by U_{x_s})
x_0	0.109	0.252	0.561	-0.155	5
x_1	0.022	0.109	0.530	-0.211	13
x_2	0.243	0.265	0.616	-0.176	8
x_3	**0.239**	**0.406**	**0.638**	**-0.116**	**3**
x_4	0.275	0.328	0.671	-0.172	7
x_5	0.198	0.253	0.632	-0.190	11
x_6	0.109	0.222	0.573	-0.176	8
$x_{0,6}$	0.049	0.514	0.875	-0.181	10
$x_{2,3}$	**0.014**	**0.783**	**0.840**	**-0.029**	**1**
$x_{2,4}$	0.075	0.549	0.888	-0.170	6
$x_{3,4}$	0.276	0.768	1.007	-0.120	4
$x_{2,3,4}$	**0.022**	**1.030**	**1.203**	**-0.087**	**2**
$x_{2,3,6}$	0.029	0.790	1.206	-0.208	12

An important aspect of the utility scores is that the score for a scenario consisting of multiple targets is not the sum of the individual utility scores for the individual targets. For example, scenario $x_{0,6}$ has a utility score of -0.181 and consists of the targets v_0 and v_6. The utility score for scenario x_0 (targeting v_0) is -0.155 and for scenario x_6 (targeting v_6) is -0.176. The sum of these two scores is -0.331. The consequence of this observation is that the utility score for each possible scenario must be calculated individually.

One important observation is that high scoring individual targets, like v_3 above, tend to be a target in high scoring multitarget scenarios. Scenario x_3, consisting of target v_3, is the highest scoring single-target scenario. The three highest scoring multitarget scenarios are $x_{2,3}$, $x_{2,3,4}$, and $x_{3,4}$, and all three targets contain the target v_3. This idea

that high scoring multitarget scenarios consist of high scoring single targets will be leveraged in the next section to develop a screening function.

3.4 Screening Functions

The challenge of determining the most advantageous insertion scenarios for a network is the mere number of possible scenarios. Table 5 shows the number of possible insertion scenarios based on a network's size and the number of targets. The growth experienced by the number of possible insertion scenarios follows a power law distribution as network size and number of targets per scenario increases (Figure 7). For this research, scenarios are limited to a maximum of three targets, but four-target scenarios are shown in Table 5 and Figure 7 to highlight how quickly the number of possible scenarios exceeds one billion.

Table 5. Number of Insertion Scenarios per Network Size & Number of Targets

Network Size	Number of Insertion Scenarios				Total Number of Scenarios
	One Target	Two Targets	Three Targets	Four Targets	
25	25	300	2,300	12,650	15,275
50	50	1,225	19,600	230,300	251,175
100	100	4,950	161,700	3,921,225	4,087,975
200	200	19,900	1,313,400	64,684,950	66,018,450
400	400	79,800	10,586,800	1,050,739,900	1,061,406,900
800	800	319,600	85,013,600	16,938,959,800	17,024,293,800

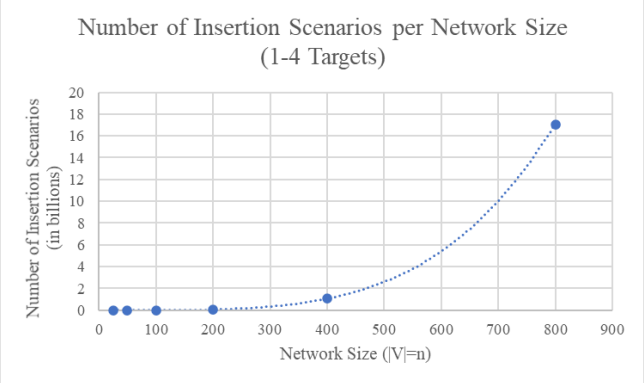

Figure 7. Growth in the Number of Scenarios Follows a Power Law Distribution

To reduce the solution space, specifically for network sizes greater than 150 nodes, this research developed three screening functions to reduce the total number of scenarios and therefore the solution space that a heuristic will search. This research utilizes the following three screening functions: neighbor-access, one-target utility score, and SNA measure.

3.4.1 Neighbor-Access Screening

Neighbor-access is a concept developed during this research and is derived from the concept of dispersion, discussed in Chapter II. Neighbor-access is the number of members within the network with a shortest path distance of 2 from the agent, post-insertion. These are the members of the network that are directly connected to the targets, and therefore, the agent would gain access to these members through the targets. For single-target scenarios, the neighbor-access score is equivalent to the degree of the target. The example network will be used next to further explain the calculation of this score.

In the example network displayed in Figure 8. insertion scenario x_0 has a neighbor access score of three. By establishing a relationship with v_0 the agent will have access to v_1, v_5, and v_6. These three members all have a shortest path distance to the agent equal to 2.

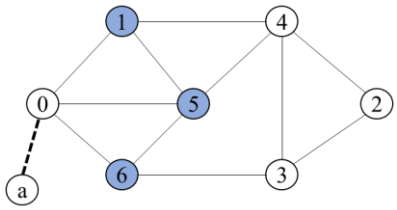

Figure 8. In Scenario x_0, the Agent Gains Access to the Members Highlighted in Blue by Targeting v_0 Resulting in a Neighbor-Access Score of 3

There is no correlation between the number of targets in an insertion scenario and the scenario's neighbor-access score. In Figure 9, the scenario consists of the agent developing relationships with v_2 and v_3. Targeting v_2 results in access to v_3 and v_4, and a relationship with v_3 provides access to v_2, v_4, and v_6. Utilizing a shortest path distance of 2 ensures that redundant neighbors of targets, and the targets themselves, are not counted toward the neighbor access score. Since v_4 and v_6 are the only members with a shortest path distance from the agent equal to 2, they are the only members counted in the neighbor-access score, resulting in a neighbor access score of 2. This is one less than the above scenario where there is one target, but the neighbor-access score is three.

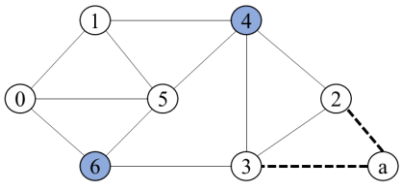

Figure 9. In Scenario $x_{2,3}$, the Agent Gains Access to the Members Highlighted in Blue by Targeting v_2 & v_3 Resulting in a Neighbor-Access Score of 2

In initial testing, it was observed that scenarios with high utility scores tend to have high neighbor-access scores, and therefore screening out scenarios with a low neighbor-access score is a promising screening method for this research. This relationship between utility scores and neighbor-access scores is understandable: scenarios with higher neighbor-access scores have a higher number of connections which can increase the targets' eigenvector and betweenness centralities, both of which are used to calculate the scenario's benefit score.

The neighbor-access screening algorithm consists of three steps. The first step is to calculate the neighbor-access score for all possible scenarios. Then, within each scenario type (one-target, two-target, etc.), the scenarios are ordered by their neighbor-access score, largest to smallest. Finally, for each scenario type, the top β scenarios are selected to be passed to the heuristic, where β is a hyperparameter β for this screening technique. Figure 10 is the flowchart of the neighbor-access screening process.

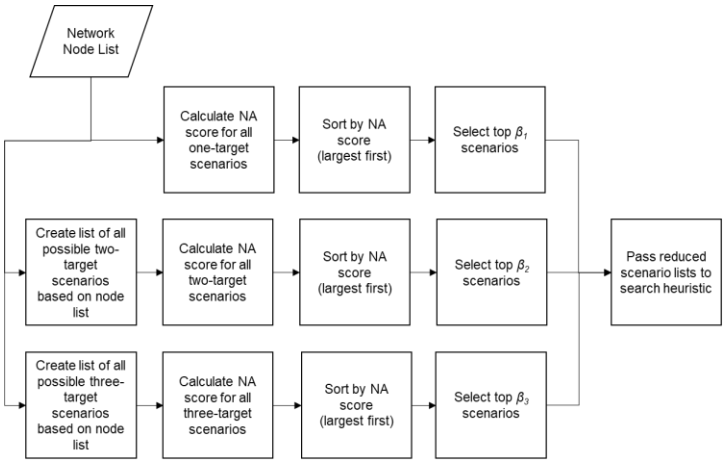

Figure 10. Flowchart of the Neighbor-Access Screening Process

For this research the most targets per scenario analyzed is three. Therefore, three values for β are used: β_1, β_2, and β_3. These separate hyperparameters allow the analyst to select a different number of top scenarios from the one-target, two-target, and three-target lists. This allows for the same portion of scenario type to be analyzed by the search heuristic. For example, analyzing a 50-node network using a single β of 15 results in 15 one-target, 15 two-target, and 15 three-target scenarios being analyzed by the search heuristic. This is 30%, 1%, and 0.076% of the possible one-target, two-target, and three-target scenarios, respectively. Using three values for β allows for an equal portion of each scenario type to be analyzed. Furthermore, allowing three values for β allows for

comparison with the other screening functions to ensure the same number of scenarios are available for the respective heuristic to evaluate.

The output of the neighbor-access screening function consists of a list of scenarios for each scenario type. These lists are then passed to a search heuristic for analysis of the solution space with respect to the utility function.

3.4.2 Utility Score Screening

As discussed in Section 3.3, early testing showed that high scoring, multitarget scenarios tend to consist of targets that individually scored high as a one-target scenario. This research leverages this information to develop its second screening function called utility score screening.

Utility score screening consists of three steps. The first step is calculating the utility score for all single-target scenarios. Then, the targets are ranked from largest to smallest based on their respective single-target utility scores. Finally, the top μ targets are selected, where μ is the hyperparameter for this screening function and determines the number of top targets that are selected. Figure 11 shows the flowchart for this process.

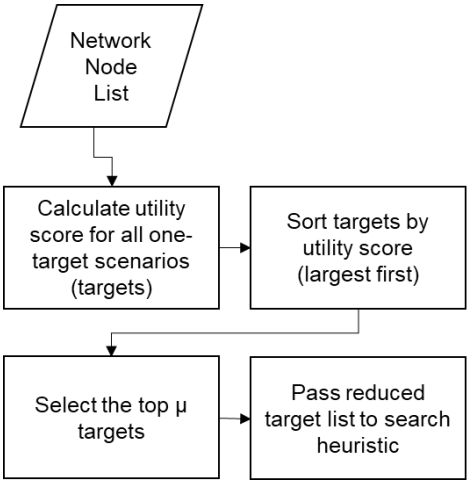

Figure 11. Flowchart of Utility-Score Screening Process

The output of the utility score screening function is a ranked set of targets. This set of targets is then passed to the search heuristic. The search heuristic will be discussed in Section 3.5. Table 6 shows the example network's single-target scenarios ranked by utility score. For utility score screening, if μ equals three, then the screening function will output the following set of targets: $\{v_3, v_0, v_4\}$. By reducing the target list from seven to three, the number of possible multitarget scenarios reduces from fifty-six to four, reducing the solution space for the search heuristic. Setting μ too low could result in a small target set that creates a reduced solution space and excludes many of the top-scoring scenarios. A μ too large can increase computation time without a guaranteed increase in benefit.

Table 6. Utility Scores for Single-Target Scenarios in Example Network

Scenario	Utility Score U_{x_s}	Rank (by U_{x_s})
x_3	-0.116	1
x_0	-0.155	2
x_4	-0.172	3
x_6	-0.176	4
x_2	-0.176	4
x_5	-0.190	6
x_1	-0.211	7

3.4.3 SNA Measure Screening

SNA measure screening follows the same process and uses the same hyperparameter (μ) as utility score screening, except that a specified SNA measure is used to rank the individual targets instead of the utility score to rank scenarios. For measure screening, the specified measure is calculated prior to node insertion. This is different from the SNA measures used to calculate the utility score, because those measures are mostly calculated after agent insertion. For this research, eigenvector

centrality is utilized as the screening measure, as initial tests showed it performed better than betweenness and closeness centralities. Table 7 displays the example network's individual targets ranked by their initial eigenvector centralities. With a μ of three, the reduced target set would be $\{v_5, v_4, v_1\}$. Like utility score screening, the reduced target set is passed to the search heuristic.

Table 7. Initial Eigenvector Centralities of Targets in Example Network

Target	Eigenvector Centrality (c_i^e)	Rank (by c_i^e)
v_5	0.481	1
v_4	0.435	2
v_1	0.396	3
v_0	0.379	4
v_6	0.359	5
v_3	0.313	6
v_2	0.229	7

3.5 Search Heuristic

This study uses two similar heuristics to search the reduced solution space provided by the three screening functions. Two heuristics are required since neighbor-access screening outputs lists of pre-built scenarios, whereas utility-score and measure screenings pass a single list of targets, which requires the heuristic to build each scenario prior to evaluation.

For utility screening and measure screening function, a construction heuristic is used. Leveraging the observation that high-scoring, multitarget scenarios tend to be comprised of individually high-scoring targets, the construction heuristic builds insertion scenarios starting with the highest scoring target based upon the screening function. The

heuristic moves about the solution space by iterating through the ordered set provided by the screening function, creating and evaluating one scenario per iteration.

The heuristic performs two runs per target list. The first run builds and evaluates the two-target scenarios, and the second run builds and evaluates the three-target scenarios. For the measure screening function, there is a third run that evaluates the single-target scenarios contained within the target set provided by the screening function. This step is not required for the utility-score screening target set as they are already evaluated by the screening function.

Performing the measure-screening function with $\mu = 5$ on the example network provides the heuristic with the following target set: $\{v_5, v_4, v_1, v_0\}$. The first run of the heuristic will build and evaluate the following two-target scenarios: (v_5, v_4), (v_5, v_1), (v_5, v_0), (v_4, v_1), (v_4, v_1), and (v_1, v_0). For the second run, the heuristic will build and evaluate the following three-target scenarios: (v_5, v_4, v_1), (v_5, v_4, v_0), and (v_4, v_1, v_0).

When applying the heuristic to the three lists provided by the neighbor-access screening function, the heuristic loses its "construction" characteristic, as the scenarios are previously constructed within the neighbor-access screening function. The heuristic still performs a search of the solution space, iterating down the two- and three-target scenario lists from highest neighbor-access score to the lowest. During each iteration, the heuristic calculates the utility score for the scenario and compares it to the current best utility score found so far.

Both forms of the heuristic utilize two hyperparameters. The first, *q*, controls the maximum number of iterations the heuristic will perform. The second, *t*, serves as the stagnation criteria that will halt the heuristic if a better solution is not found within *t*

iterations. Due to there being different numbers of possible two- and three-target solutions, the analyst can input a different value for *q* and *t* for the two-target run and the three-target run. If the same value of *q* is used for both runs, a smaller portion of the possible three-target scenarios are evaluated when compared to the portion of two-target scenarios evaluated. Allowing two values for *q* and *t* allows the analyst to prevent the above from occurring. The maximum number of iterations used for the two-target scenario run of the heuristic should not exceed the number of possible two-target scenarios based on the inputted target set. The same is true for the maximum number of iterations for the three-target scenario run. When applying the heuristic to the neighbor-access screening lists, the maximum number of iterations should not exceed the number of top two- and three-target scenarios kept by the neighbor-access screening function.

3.6 Case Study

In this section, the previously discussed methodology is applied to the September 11th Terrorist Network depicted in Figure 12. This network was developed from the information provided in *The 9/11 Commission Report* [39]. The undirected network consists of 69 individuals and 271 edges, resulting in a network density of 0.115. The edges represent various forms of connection between members and are not weighted. The average degree within the network is 7.9, the maximum degree is 25, the minimum is one, and median degree is six.

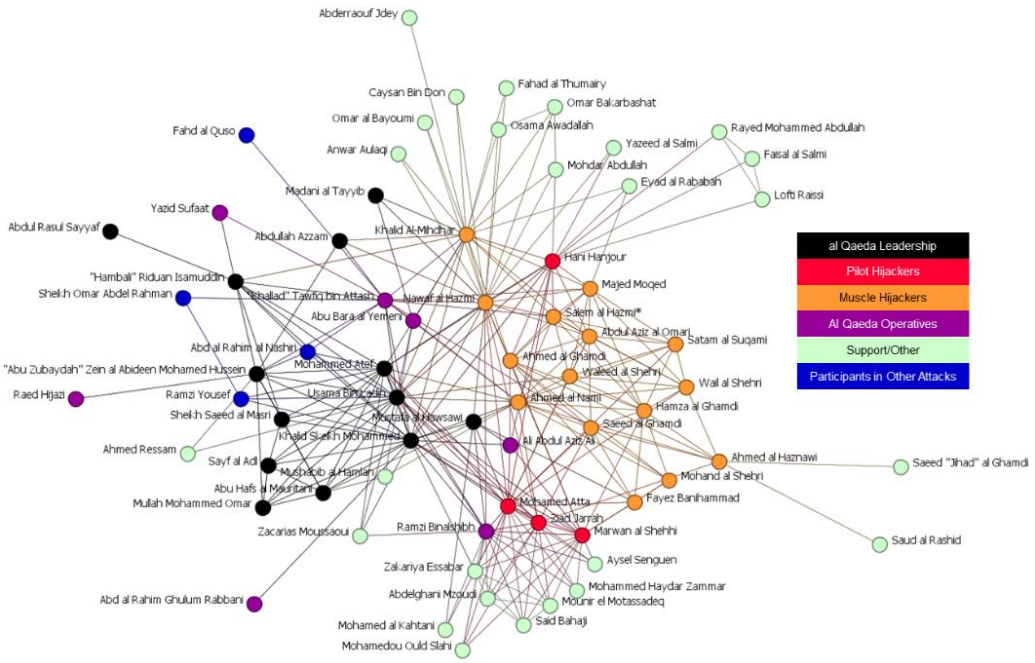

Figure 12. Network Depiction of the Terrorists Involved in the September 11th Attacks, Colored by Member's Role.

With 69 members in the network, there are 54,809 possible insertion scenarios when evaluating one-, two-, and three-target scenarios. Table 8 displays the 10 most advantageous scenarios (greatest utility score) when using an arbitrary cost of 0.2 and equal weights.

Table 8. Top 10 Scenarios for the 9/11 Network (Cost = 0.2, Equal Weights)

Rank	Scenario	Targets	B_{x_s}	R_{x_s}	U_{x_s}
1	x_3	N. al Hazmi	0.234	0.390	-0.078
2	x_{31}	K.S. Mohammed	0.226	0.391	-0.083
3	$x_{3,31}$	N. al Hazmi; K.S. Mohammed	0.445	0.611	-0.083
4	$x_{3,31,50}$	N. al Hazmi; K.S. Mohammed; U. Bin Ladin	0.645	0.817	-0.086
5	$x_{31,50}$	K.S. Mohammed; U. Bin Ladin	0.425	0.597	-0.086
6	x_{50}	U. Bin Ladin	0.213	0.390	-0.088
7	$x_{3,50}$	N. al Hazmi; U. Bin Ladin	0.433	0.611	-0.089
8	x_{14}	M. Atta	0.203	0.385	-0.091
9	$x_{3,14}$	N. al Hazmi; M. Atta	0.423	0.610	-0.094
10	$x_{1,3}$	K. al Mihdhar; N. al Hazmi	0.411	0.602	-0.096

Table 9 displays the results of both utility screening and measure screening when applied to the 9/11 network for μ=15. The target lists created by both screening functions contain the same targets except for two: v_4 and v_{46}. The rankings of the targets within the list are also different. With a target list of size 15, there are 15 one-target, 105 two-target, and 455 three-target scenarios, for a total of 575 possible scenarios to evaluate with the search heuristic.

Table 9. Results from Utility Score and Measure Screenings on 9/11 Network (μ=15)

Rank (μ=15)	Utility-Score Screening	Measure Screening
	Target	Target
1	v_3	v_{31}
2	v_{31}	v_{50}
3	v_{50}	v_{14}
4	v_{14}	v_3
5	v_1	v_{36}
6	v_2	v_{12}
7	v_{12}	v_1
8	v_{36}	v_{13}
9	v_{37}	v_{25}
10	v_5	v_2
11	v_{13}	v_5
12	v_{25}	v_{37}
13	v_6	v_{15}
14	v_{46}	v_4
15	v_{15}	v_6

For the application of the heuristic to the target lists provided by the screening functions, the maximum number of iterations (q) allowed is 105 for two-target scenarios and 455 for three-target scenarios. The stagnation criteria (t) is set at roughly 30% of the maximum allowed iterations, resulting in a stagnation criterion of 30 and 131 for two-target and three-target scenarios, respectively.

When running the heuristic on the utility screening target list, 31 iterations are performed for two-target scenarios and 132 for three-target scenarios before the heuristic

meets the stagnation criteria. The average computational time for both the utility screening and the heuristic across 10 runs is 0.787 seconds. Table 10 shows which iteration the heuristic discovered the highest scoring scenario per scenario type. The early discovery of the highest scoring scenario highlights the observation that high scoring scenarios consist of individual targets that are scored highly by the screening criteria. Using the utility screening target list, the first two-target and three-target scenarios evaluated by the heuristic are the highest scoring for their respective scenario type.

Table 10. Iteration When Heuristic Evaluates the Highest Scoring Scenario

Iteration	Utility Screening Target List		Measure Screening Target List	
	Two-Target Scenarios	Three-Target Scenarios	Two-Target Scenarios	Three-Target Scenarios
1	$x_{3,31}$	$x_{3,31,50}$	$x_{31,50}$	$x_{31,50,14}$
2			$x_{31,14}$	$x_{31,50,3}$
3			$x_{31,3}$	

Utilizing the measure screening target list, the heuristic does not evaluate the highest scoring scenarios until iteration 3 and 2 for two- and three-target scenarios, respectively. This results in a total of 33 iterations for two-target scenarios and 133 iterations for three-target scenarios until the stagnation criteria is met. Across 10 runs, the processing time for the measure screening function and heuristic is 0.617 seconds.

The shorter processing time of measure screening when compared to utility score screening is a result of the screening functions and not the construction heuristic. Utility score screening requires the utility score, and all its required measures, to be calculated once for each node in the network. Therefore, betweenness, eigenvector, and closeness

centralities are each calculated 69 times for the 9/11 instance. Compare this to the one-time calculation of eigenvector centrality required for measure screening.

When applying neighbor-access screening to the 9/11 Network the following values for β are used: $\beta_1 = 15$, $\beta_2 = 105$, $\beta_3 = 455$. These values align with the total number of one-, two-, and three-target scenarios that are possible from the utility-score and measure screening functions when μ=15. Table 11 shows the top ten scenarios for each scenario type based on the outputs of the neighbor-screening function. The highest scoring two- and three-target scenarios are highlighted in grey.

Table 11. Neighbor-Access Screening Output per Scenario Type (Top 10)

Rank (By Neighbor-Access Score)	One-Target Scenarios	Two-Target Scenarios	Three-Target Scenarios
1	x_3	$x_{1,14}$	$x_{1,14,31}$
2	x_{14}	$x_{1,31}$	$x_{1,14,50}$
3	x_{31}	$x_{3,14}$	$x_{1,14,36}$
4	x_1	$x_{1,13}$	$x_{3,14,50}$
5	x_{50}	$x_{1,50}$	***$x_{3,14,31}$***
6	x_{13}	***$x_{3,31}$***	$x_{1,13,31}$
7	x_{36}	$x_{3,50}$	$x_{1,13,50}$
8	x_2	$x_{14,50}$	$x_{1,14,44}$
9	x_5	$x_{1,12}$	$x_{1,14,46}$
10	x_{12}	$x_{1,25}$	$x_{3,13,50}$

The heuristic completes 36 iterations searching the two-target scenarios list and 136 iterations on the three-target list before meeting the stagnation criteria. The average processing time across the ten runs is 0.831 seconds. Reviewing Table 11, it can be observed that the heuristic finds the highest scoring two-target scenario on the sixth iteration. This scenario is also the global optima for all two-target scenarios. For three-target scenarios, the heuristic finds the highest scoring scenario on iteration five. Unfortunately, this is not the global optima for three-target scenarios. Scenario $x_{3,31,50}$ is

the global optima for three-target scenarios and the heuristic would have to conduct at least 401 iterations to discover this scenario based on its placement on the scenario list provided by the neighbor-access screening.

Table 12 displays the *accuracy* of each of the screening functions and the search heuristic. Accuracies are calculated by comparing the screening heuristics' outputs with that of a "brute force" all-scenarios model that calculates the utility score for all possible scenarios. Accuracy measurements are taken for the top 1, 5, 10, 25, and 50 scenarios when compared to those found by the all-scenarios model. For example, if the screening heuristic finds 4 out of the 5 top scenarios, then its accuracy for Top 5 is 0.80.

Table 12. Accuracy of Screening Functions for 9/11 Network

Screening Function	Accuracy					CPU Time (s)
	Top 1	Top 5	Top 10	Top 25	Top 50	
Utility-Score	1.00	1.00	1.00	1.00	1.00	0.787
SNA Measure	1.00	1.00	0.90	0.92	0.92	0.617
Neighbor-Access	1.00	0.60	0.70	0.56	0.56	0.831

Reviewing the results in Table 12 reveals that for this problem instance, utility-screening is the most accurate. Measure screening maintains an accuracy of 0.90 through the Top 50. Neighbor-Access barely surpasses 0.50 for the Top 25 and 50. Accuracy of neighbor-access can be improved by increasing the stagnation criteria and the maximum number of iterations for its heuristic.

As discussed before, this research is not solely interested in finding the global optima, but instead interested in finding a specified number of top scoring insertion scenarios that can be provided to intelligence analyst for further analysis and decision. For example, the top three targets highlighted in this case study are Nawaf al Hazmi, Khalid Sheik Mohammed, and Usama Bin Laden. All three of these individuals are key

members within the 9/11 network. These targets would garner extensive intelligence information, but the risk to the insertion agent would be too great. Understanding the structure of the network provides insight into why these individuals were selected as the highest scoring targets. First, the diameter of the network is only five, meaning that the maximum number of edges between any two members within the network is five. Additionally, the average degree within the network is 7.9. The degrees of the three individuals are all over 20. These factors help to increase the target's utility scores when compared to other members of the network. To counter these effects, the weights within the utility function can be altered to emphasize different aspects in the hopes of obtaining different scenarios.

3.7 Chapter Summary

This chapter discussed the methodology used for this research. It provided a discussion on the development of the scoring method for insertion scenarios utilizing risk and benefit scoring equations, which together form the overall utility score for each scenario. Three screening functions, utility-score, measure, and neighbor-access, were developed to decrease the solution space prior to applying a search heuristic. The chapter also discussed the operations of the search heuristics used for this research. Finally, the methodology was applied to the September 11th Terrorist Network case study to illustrate this methodology in action.

IV. Analysis and Results

4.1 Chapter Overview

This chapter details the testing of the three screening heuristics and provides analyses of factors that affect heuristic performance. The chapter starts with a discussion of the testing plan. This is followed by evaluation of the effects that utility function parameters and heuristic hyperparameters have on heuristic performance. This information then informs the settings used for the overall performance evaluation of each screening heuristic. The chapter concludes with analysis of network characteristics that affect accuracy of the heuristics and an estimation of performance on larger networks.

4.2 Testing Plan Overview

For this study, testing consisted of four stages. The first two stages test the effects of the utility function parameters and the heuristic hyperparameters on the accuracy of the three screening heuristics. The information gained during stage one and two influenced the values used for the parameters and hyperparameters in stage three. Stage three testing evaluates the accuracy and computational time required for each heuristic. Stage four will evaluate network factors that affect heuristic accuracy.

For stage one and two of testing, eighteen benchmark networks were used, created using three network generators from Python's *NetworkX* [40] module: Erdős–Rényi (ER), relaxed caveman (RC), and Barabási-Albert (BA). Table 13 shows the benchmark networks used for this analysis and their respective characteristics. For all graph types, n is the number of nodes in the network and *seed* is the seed number for replication. For ER graphs, p is the probability that an edge is formed during generation, but for RC

graphs *p* is the probability that an edge is rewired to a node outside of its cluster. Additionally, for RC, *k* is the number of nodes per group and *l* is the number of groups. BA graph generation utilizes *m* the number of edges to connect from a new node to existing nodes.

Table 13. Benchmark Network Information

Network Index	Network Generator	*n*	*p*	*k*	*l*	*m*	Seed
1	BA	50	-	-	-	3	326
2	BA	50	-	-	-	6	439
3	BA	100	-	-	-	3	48
4	BA	100	-	-	-	6	35
5	BA	200	-	-	-	3	402
6	BA	200	-	-	-	6	375
7	RC	50	0.25	10	5	-	549
8	RC	50	0.25	5	10	-	733
9	RC	105	0.35	7	15	-	85
10	RC	140	0.25	20	7	-	763
11	RC	200	0.20	10	20	-	472
12	RC	200	0.20	20	10	-	460
13	ER	50	0.10	-	-	-	137
14	ER	50	0.25	-	-	-	277
15	ER	100	0.10	-	-	-	247
16	ER	100	0.25	-	-	-	145
17	ER	200	0.10	-	-	-	19
18	ER	200	0.25	-	-	-	160

Stage three of testing utilizes 18 trials, each using the same network settings as the benchmark networks, with two exceptions. First, each trial will consist of multiple runs, with each run consisting of a different randomly generated graph. Second, the utility score parameters and heuristic hyperparameters values used during stage three will be informed by the results of previous stages. Stage four will then use the outputs of stage three to determine what network factors affect the accuracy of each heuristic. More details on each stage's testing procedures are found in their respective sections.

4.3 Testing of Utility Score Parameters

Prior to evaluating the performance of the screening heuristics, the utility score's cost factor and weights are evaluated to understand their effects on the ranking of scenarios.

4.3.1 Cost Factor

As discussed in Chapter III, the cost factor is used in the utility score function to model the risk associated with developing multiple relationships within a scenario. Without the cost factor, the algorithm is biased towards the scenarios with the most targets. By simple deduction, it can be reasoned that a high cost factor can bias the algorithm towards one-target scenarios, and a low cost factor can bias the algorithm towards scenarios with the highest number of targets. The purpose of this evaluation is to determine the range at which the cost factor limits bias by the algorithm and the top scoring scenarios are a mix of scenario types. Removing the bias of the algorithm is important to ensure each scenario type is treated equally by the algorithm and to remove the advantages three-target scenarios have over two-target scenarios, and two-target scenarios over one-target scenarios, due to nature of the utility function without the cost factor.

To perform this evaluation, a bracketing technique was used to find the window of cost factor values that produced the least bias, or an even mix of one-, two-, or three-target scenarios. For each cost factor value used during bracketing, each scenario's utility score was updated, and the scenarios were sorted from largest to smallest according to their utility score. Then the number of each type of scenario in the top *n* was recorded. This results in a range of cost factors where there are a mix of scenario

types in the top *n* scenarios. The *maximum* cost factor value within this range results in a majority (greater than 95%) of one-target scenarios with one or two two-target scenarios. Cost factors greater than this value result in all one-target scenarios in the top *n*. The *minimum* cost factor value results in a majority of three-target scenarios with one or two two-target scenarios present. Values less than this range results in all three-target scenarios. The *prime* cost factor value is where the least bias and the most balanced mix of one-, two-, and three-target scenario types occurs. Table 14 shows the cost factor ranges (maximum, minimum, and prime) for the benchmark networks and the density, diameter, and average degree of each network.

Table 14. Cost Factor Ranges for Benchmark Networks

Network Index	Cost Factors			Range Length	Network Density	Network Diameter	Average Degree
	Minimum	Prime	Maximum				
1	0.090	0.240	0.390	0.300	0.12	4	5.64
2	0.090	0.195	0.300	0.210	0.22	3	10.60
3	0.120	0.245	0.370	0.250	0.06	4	5.82
4	0.070	0.185	0.300	0.230	0.11	3	11.30
5	0.090	0.250	0.350	0.260	0.03	5	5.90
6	0.060	0.180	0.300	0.240	0.06	4	11.60
7	0.090	0.155	0.230	0.140	0.18	4	9.00
8	0.170	0.240	0.350	0.180	0.08	8	4.00
9	0.070	0.135	0.210	0.140	0.07	6	6.00
10	0.047	0.054	0.065	0.018	0.14	3	19.00
11	0.050	0.080	0.120	0.070	0.05	6	9.00
12	0.040	0.048	0.065	0.025	0.10	4	19.00
13	0.110	0.185	0.280	0.170	0.10	5	4.72
14	0.080	0.115	0.150	0.070	0.26	3	12.56
15	0.060	0.080	0.120	0.060	0.10	4	10.10
16	0.045	0.058	0.085	0.040	0.26	3	25.60
17	0.045	0.057	0.070	0.025	0.10	3	20.00
18	0.030	0.034	0.043	0.013	0.25	2	50.40

Figure 13 shows the effect of changing the cost factor on utility scores per scenario type in Benchmark Network 3, which was selected to represent the effects seen across all benchmark networks. The red line indicates the utility score of the 100th ranked scenario. At a cost factor of 0.12, the top 100 scenarios consist of 4 two-target

and 96 three-target scenarios. As the cost factor increases, the mix of scenario types within the top 100 changes, with an increasing number of one-target scenarios entering the top 100 and three-target scenarios exiting. At a cost factor of 0.37, the mix has switched to 96 one-target and 4 two-target scenarios. As expected, as the cost factor increases, the utility scores of all scenarios decrease. Similar results were seen in all benchmark graphs.

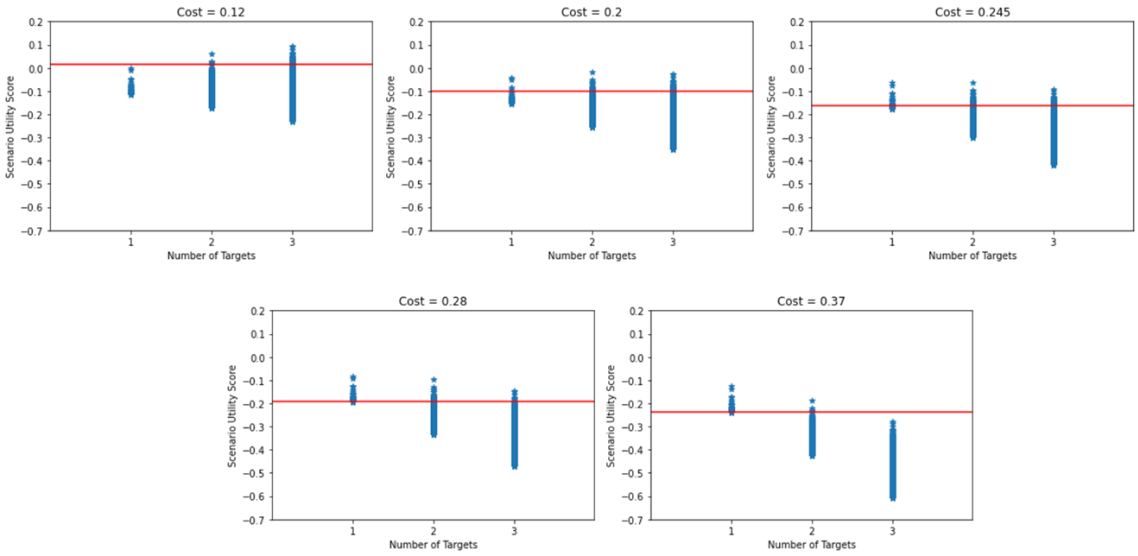

Figure 13. The Effect of Cost Factor Values on Scenario Utility Scores in Benchmark Network 3

Figure 14 shows the effects of a network's density and average degree on the prime cost factor value. In each graph, points are sized according to the number of nodes in the network. The lines connecting points, connect points representing networks of the same size. When comparing networks of the same type and size, increasing either network density or average degree results in a decreased prime cost factor value.

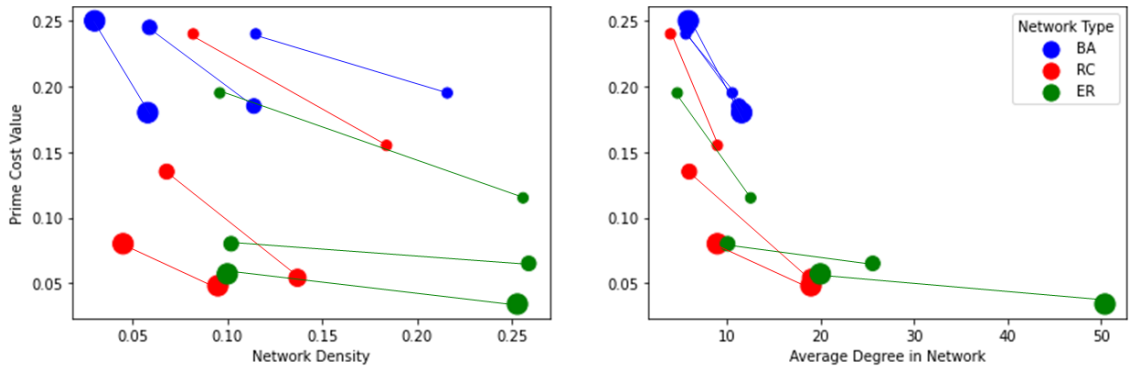

Figure 14. Effects of Network Density & Average Degree on Prime Cost Factor Value (Point Size Reflects Network Size).

Figure 14 further reveals that cost factors for BA graphs are less affected by network size than ER and RC graphs. Both ER and RC graphs have smaller cost factors values for larger networks. This could be the result of BA graphs maintaining similar network structures despite size.

After looking at the effects of different cost factors on the utility score, the screening heuristics were run using varying values of the cost factors to determine if there is any relationship between cost factor values and heuristic accuracy. Utilizing the benchmark graphs, the screening heuristics were run five times each, using one of five cost factors. The cost factors utilized were within the ranges presented in Table 14 for each respective network. The maximum, minimum, and prime cost factors in the ranges were used, along with the two midpoints of the prime and extreme cost factors in the range.

The results show that a high cost factor can have an effect on the accuracy of the heuristic, but it depends on the value of μ or β_1 and the number of top scenarios being compared. If a 100-node network is being evaluated by measure screening with a $\mu = 15$ and the top 50 scenarios are being compared to those of the all-scenarios model, the

greatest accuracy the heuristic can achieve is 0.3 (15 out of 50). The accuracy could be lower depending on which one-target scenarios are kept by the screening mechanism. With a high cost factor, the top 100 scenarios found by the all-scenarios model will be all one-target scenarios. With a $\mu = 15$ the heuristic will only evaluate 15 one-target scenarios, resulting in the top 100 scenarios consisting of only 15 one-target scenarios and the remainder being either two- or three-target scenarios. Therefore, when comparing the top 50 scenarios from the heuristic with those from the all-scenarios model, only 15 of the 50 scenarios will match. The accuracy will increase as μ increases or the number of scenarios being compared decreases.

For the utility-score screening heuristic, this is not the case. Due to the nature of the screening function, the utility scores for all one-target scenarios are calculated during the screening step. This is prior to creating a reduced target list that is passed to the heuristic. The utility scores for the one-target scenarios not selected by the screening function are kept and ranked with all other scenarios evaluated by the heuristic. Therefore, with a network size greater than 50, if all scenarios in the top 50 are one-target scenarios, the utility-screening heuristic will have an accuracy of 1.0.

In summary, for a given network instance, there is a range of cost factor values that provide a mix of one-, two-, and three-target scenarios. Values greater than this range bias one-target scenarios, resulting in the top *n* scenarios being all one-target scenarios, while values less than this range favor three-target scenarios and result in the top *n* scenarios consisting of all three-target scenarios. Both density and average degree all affect the prime cost factor value for a network.

4.3.2 Weights

To evaluate the effect of the various weights within the utility function, a process similar to evaluating the cost factor effects is used. Utilizing the benchmark networks, the utility scores were first calculated using all weights equal to 0.5. Then each pair of weights (w_B & w_R, w_e & w_b, w_c & w_γ) were tested with values of 0.75 and 0.25, in both configurations. For instance, w_e was set to 0.75 and w_b set to 0.25. In the next iteration, the values were switched, so $w_e = 0.25$ and $w_b = 0.75$. Then w_c and w_γ were evaluated, followed by w_B and w_R. During each evaluation, the mix of scenario types in the top *n* were calculated, in addition to the specific scenarios that were within the top 10. The following example discusses specifically Benchmark Network 9, but similar results were observed for all benchmark networks.

Figure 15 displays the effect of changing a specific weight on the mix of scenario types within the top 100 scenarios for Benchmark Network 9. The red data points represent the top 10 scenarios when all weights are equal to 0.5 and continue to represent the same scenarios throughout all other graphs. This allows the reader to see how the top scenarios change with the weights. The green horizontal line represents the utility score of the 100th ranked scenario given the specific weight modification noted in the individual graph's title.

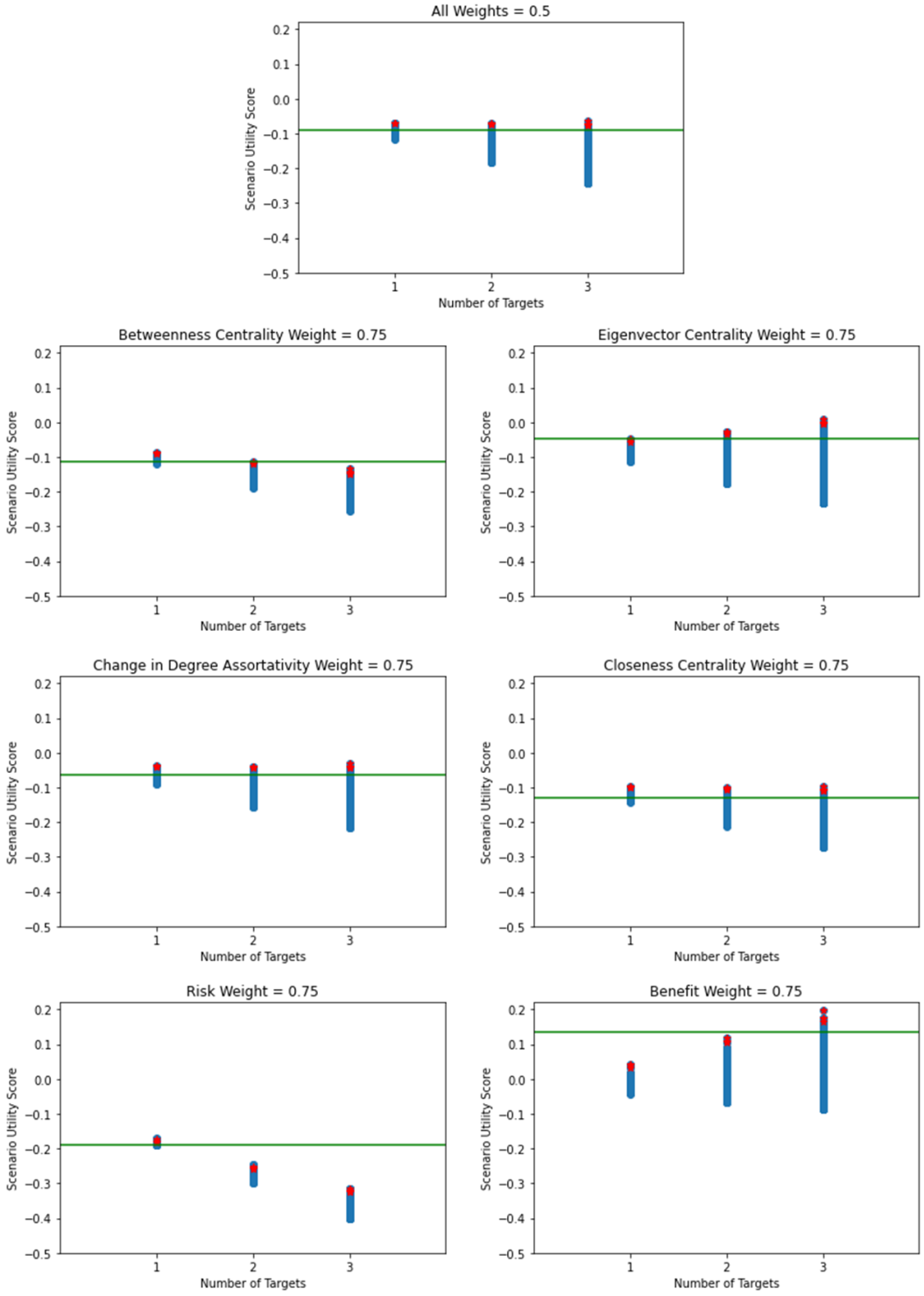

Figure 15. Changing the Weights within the Utility Function can also Bias the Algorithm, Similar to the Cost Factor

For two of the three sets of weights (w_B & w_R, w_e & w_b) in Benchmark Network 9, increasing the value of one of the weights above 0.5 biases the algorithm towards one-target scenarios and by increasing the other weight value above 0.5, causes the algorithm to be biased toward three-target scenarios. In the graphs above, when $w_e = 0.75$ and $w_b = 0.25$, the top 100 scenarios consist of mainly three-target and a few two-target scenarios. Conversely, when $w_b = 0.75$ and $w_e = 0.25$, one-target scenarios are favored. The same occurs when evaluating w_B and w_R.

Changing the values w_c and w_γ has less of an effect on the mix of scenario types within the top 100. When weights are equal and the cost factor is 0.135, there are 37, 22, and 41 one-, two-, and three-target scenarios within the top 100, respectively. When $w_c = 0.75$, the mix becomes 48, 22, and 30. For $w_\gamma = 0.75$, the mix is 26, 22, and 52. Changing these two weights does slightly bias the algorithm to a certain scenario type, but not to the extent of the other weight sets, where the top 100 mix is either almost all one-target scenarios or almost all three-target scenarios. Furthermore, when all other weights are set at 0.5 each, there is no value between zero and one where w_c, and conversely w_γ, forces the scenarios in the top 100 to be either almost all one-target or almost all three-target scenarios like the other two sets of weights.

When comparing the top 10 scenarios after each weight change, the top scenarios consisted of the same targets, with one exception. When $w_R = 0.75$, scenarios consisting of different targets entered the top 10. This can be seen in Figure 15 by tracking the movement of the red data points. Figure 16 displays zoomed in versions of certain graphs from Figure 15. Please note that each graph in Figure 16 has a different vertical scale.

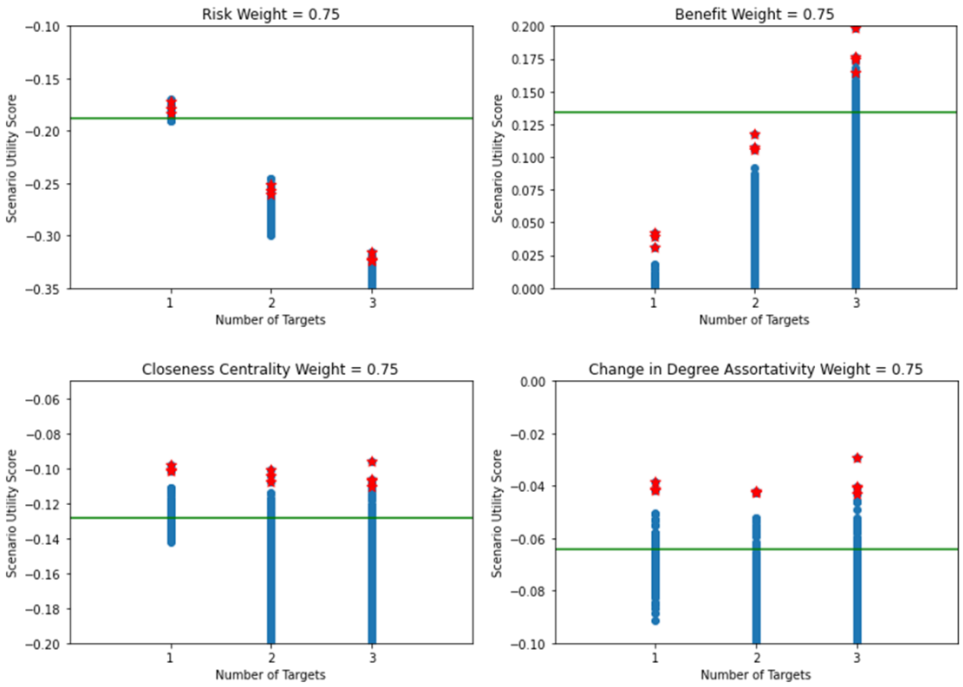

Figure 16. Increased Fidelity of Certain Graphs from Figure 9 to Highlight the Unique Effect when Risk Weight $w_R = 0.75$ (Top Left)

Like cost factor values, there is a range of values for each weight where a mix of scenario types exist. For Benchmark Network 9, the range for w_e is from 0.3 to 0.75. Values below or above this range will result in either all one-target scenarios or all three-target scenarios, respectively. Conversely, the range for w_b is 0.7 to 0.25. For w_c and w_γ, there is no value between zero and one that causes the top 100 scenarios to be either all one-target or all three-target scenarios. Network size does influence the length of the weight ranges. Smaller networks tend to have a wider range of weight values where scenario type mixing occurs when compared to larger networks where the range is smaller.

After reviewing the effects of both the weights and the cost factor on the mix of scenario types within the top *n* scenarios, it is recommended that the weights are established before the cost factor. The analyst should first establish the weights to

emphasize specific measures within the utility function, then set the cost factor to tune the scenario type mix within the desired top range of scenarios.

4.4 Hyperparameters

The purpose of testing the hyperparameters is to determine a balance between processing time and accuracy of the heuristics. It is proposed that increasing the values of all the hyperparameters would increase the accuracy of the screening heuristics. Keeping more targets after screening increases the likelihood that a top scenario is discovered by the heuristic. Matched with an increased *q* (stagnation criteria) and *t* (maximum iterations), it is expected that the heuristic will have an increased accuracy. Yet, this increased accuracy would result in increased processing time since more iterations will be performed. Therefore, this research tests the hyperparameters to find the respective values that provide a suitable balance between accuracy and processing time.

4.4.1 μ and β

The hyperparameters μ and β determine the number of targets or scenarios that are passed from the screening function to the heuristic. μ is used by both utility-score and measure screening functions. It determines the number of top targets, per the screening criteria, that are passed to the heuristic. β is used by neighbor-access screening and consists of three values: β_1, β_2, and β_3, which determine the number of top one-, two-, and three-target scenarios, respectively, that are passed to the heuristic.

To test μ and β, multiple iterations of the heuristics were conducted at increasing values based on specified percentages of *n*, the total number of targets in a network. For the benchmark networks, μ and β_1 were varied between 15% and 50% of *n* in steps of

5%. To ensure consistency between the three screening heuristics, β_2 and β_3 were set to the total number of two-target and three-target scenarios that can be created using μ targets. This will allow neighbor-access screening to produce the same number of two- and three-target scenarios as that of utility-score and measure screening. This is important, as during the testing of μ and β, both *q* and *t* will be set equal to the maximum number of possible two- and three- target scenarios based on the values of μ and β.

The outcomes from testing μ and β serves two purposes. One is to understand how changing their values affect the accuracy of the heuristics. The second is to determine a value for μ and β for follow-on testing of the stagnation criteria and final heuristic performance. The percentage of *n* at which all three heuristics achieve a 0.90 accuracy will be the selected values for μ and β moving forward.

Figure 17 shows the average accuracy of each heuristic across all benchmark networks when μ and β are set to a given percentage of *n*. As suspected, as μ and β increase in value, their respective screening heuristics increase in accuracy. Neighbor-access' (NA) lower accuracy is due to its performance when dealing with networks generated by the RC technique. This will be discussed later in the chapter. Removing the performance on the RC networks, NA's average accuracy increases from 0.71 to 0.90 at 25% and 0.83 to 1.00 at 50%.

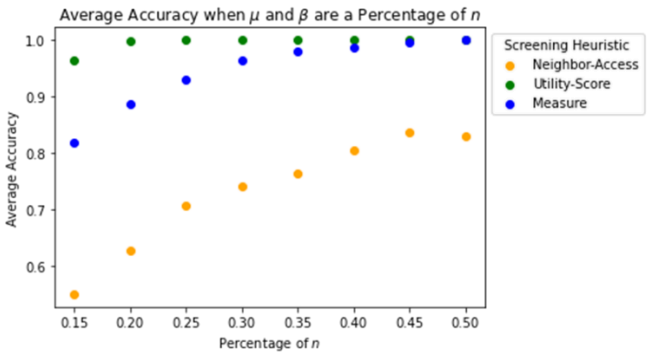

Figure 17. As μ and β Increases, the Average Accuracy of the Heuristic Increases

As mentioned previously, the percentage of n at which all three heuristics achieve 0.90 accuracy will be selected as the value for μ and β for future testing. Since NA screening does not achieve 0.90 within the selected range of percentages due to its performance with RC networks, the point where NA achieves 0.90 accuracy without the RC networks is used. This results in 25% of n as the value of μ and β in future testing.

4.4.2 Stagnation Criteria

To test the stagnation criteria for the screening heuristics, a similar approach to the testing of μ and β was used. All three screening heuristics were run on all eighteen benchmark networks using the network's prime cost factor, equal weights, and μ and β_1 equal to 25% of n. β_2 and β_3 were set to the total number of possible two-target and three-target scenarios based on the values of μ and β_1. For the stagnation criteria (q), the two-target search and three-target search stagnation criteria were both set at a percentage of the total number of possible two- and three-target scenarios based on μ and β values. The percentages ranged from 10% to 60% in 5% increments, resulting in 11 runs per benchmark network.

Figure 18 shows the results of the stagnation criteria tests. Again, both utility-score and measure screening perform well, with both achieving a 0.90 average accuracy

at stagnation criteria equal to 15% of possible scenarios. Again, NA performs poorly on the RC networks. Removing these benchmark networks from the average accuracy calculation results in a 0.90 accuracy average for NA occurring at a stagnation criteria of 50%.

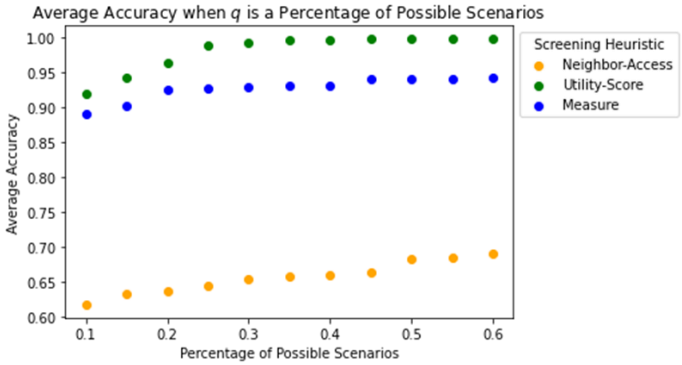

Figure 18. As *t* (Set as the Percentage of Possible Scenarios) Increases, there is Only a Slight Increase in Heuristic Accuracy.

Like μ and β, the goal with the stagnation criteria testing is to find the lowest percentage where all screening heuristics achieve an average accuracy of 0.90 across all benchmark networks. Unfortunately, similar to μ and β testing, this does not occur within the specified interval for testing, due in part to NA's performance on RC benchmark networks. To achieve the desired 0.90 accuracy, the RC networks were removed from the average calculation, resulting in a 0.90 average accuracy occurring at 50%. At a stagnation criteria of 50% of possible two- and three-target scenarios, there is still a large computational time requirement for larger graphs. Reducing the desired average accuracy level from 0.90 to 0.85 allows for a further reduction of the stagnation criteria from 50% to 25%, which will be used for the performance testing of the screening heuristics.

4.5 Screening Heuristic Performance Testing

To test the performance of each screening heuristic, 18 trials were conducted on networks with similar parameters as the benchmark networks. Table 15 contains the values for the network generation parameters and the heuristic hyperparameters for each trial. The prime cost factor values from Table 14 were used for each respective graph, and all weights were equal to 0.5. Trials were conducted on a Dell Inspiron 5515 Laptop with 16 GB of RAM and an AMD Ryzen 7 5700U (1.8GHz) processor. The Python code is provided in Appendix A.

Table 15. Trial Parameters

Trial	Network	n	p	k	l	m	μ	$\beta_1, \beta_2, \beta_3$	q	t	Runs
1	ER	50	0.10	-	-	-	13	(13, 78, 286)	(20, 72)	(78, 286)	100
2	ER	50	0.25	-	-	-	13	(13, 78, 286)	(20, 72)	(78, 286)	100
3	ER	100	0.10	-	-	-	25	(25, 300, 2300)	(75, 575)	(300, 2300)	100
4	ER	100	0.25	-	-	-	25	(25, 300, 2300)	(75, 575)	(300, 2300)	100
5	ER	200	0.10	-	-	-	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20
6	ER	200	0.25	-	-	-	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20
7	RC	50	0.25	10	5	-	13	(13, 78, 286)	(20, 72)	(78, 286)	100
8	RC	50	0.25	5	10	-	13	(13, 78, 286)	(20, 72)	(78, 286)	100
9	RC	105	0.35	7	15	-	27	(27, 351, 2925)	(88, 732)	(351, 2925)	100
10	RC	140	0.25	20	7	-	35	(35, 595, 6545)	(149, 1637)	(595, 6545)	80
11	RC	200	0.20	10	20	-	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20
12	RC	200	0.20	20	10	-	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20
13	BA	50	-	-	-	3	13	(13, 78, 286)	(20, 72)	(78, 286)	100
14	BA	50	-	-	-	6	13	(13, 78, 286)	(20, 72)	(78, 286)	100
15	BA	100	-	-	-	3	25	(25, 300, 2300)	(75, 575)	(300, 2300)	100
16	BA	100	-	-	-	6	25	(25, 300, 2300)	(75, 575)	(300, 2300)	100
17	BA	200	-	-	-	3	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20
18	BA	200	-	-	-	6	50	(50, 1225, 19600)	(307, 4900)	(1225, 19600)	20

For each trial, multiple runs were conducted. Each run consisted of a different randomly generated network based on the trial's specific parameters in Table 15. For each run the utility scores for all scenarios in the network were calculated using the "brute force" all-scenarios model. The three screening heuristics were run on the network using the hyperparameters listed for the respective trial in Table 15. Finally, the

heuristics' top 1, 5, 10, and 25 results were compared to that of the all-scenario's model. The running of the all-scenarios model was the major time driver for each run. For instance, Trial 12 averaged 2.64 hours of processing time per run to execute each run's all-scenarios model.

The initial goal for each trial was to conduct 100 runs. Due to time constraints, this was not achievable for the larger networks. For the 200-node network trials, only 20 runs per trial were conducted. For Trial 10, 80 runs were conducted. For all but three trials (6, 11, and 12), based on an error criteria of $\varepsilon = 0.05$, the number of runs conducted per trial were sufficient to achieve a 95% confidence interval based on

$$R \geq \left(\frac{t_{\alpha/2,R_0-1} S_0}{\varepsilon}\right)^2 \tag{4.1}$$

where $t_{\alpha/2,R-1}$ is the Student's t-statistic, α is the significance level ($\alpha = 0.05$), R is the number of initial replications, and S_0 is the standard deviation of the runs [41]. For the three trials that did not meet the minimum required number of runs, only one heuristic in each trial required the additional runs. For Trial 11 and Trial 6, NA required more than 20 runs, and for Trial 12, measure screening (MS) required more than 20 runs.

The following sections discuss the results of the screening heuristics' performance with respect to accuracy and computational time.

4.5.1 Accuracy Results

Accuracy of the heuristics were calculated by comparing the top 1, 5, 10, and 25 scenarios as determined by the heuristic and comparing those to the actual top 1, 5, 10, and 25 scenarios as determined by the "brute force" all-scenarios model. Accuracy is the number of scenarios that are present in both the heuristic's top scenarios list and the all-

scenarios model's top scenario list. This number is then divided by the top number of scenarios being compared (1, 5, 10, or 25) to give the heuristic's accuracy. For each trial, the runs' accuracies were averaged, and the 95% confidence intervals (CI) were determined using a significance level of 0.05 and the Student's *t*-test statistic. Table 16 displays the average accuracy for the top 25 and 95% CI for each screening heuristic during each trial. Grey highlight indicates when a screening heuristic has a significantly higher average accuracy, based on the *t*-test, than the other two heuristics for that trial.

Table 16. Average Accuracy with 95% CI per Trial and Screening Heuristic Considering Top 25 Scenarios (Grey Highlight Indicates Significantly More Accurate Heuristic During Trial)

Trial	*n*	Average Density	Neighbor-Access Accuracy (95% Confidence Interval)			Utility-Score Accuracy (95% Confidence Interval)			SNA Measure Accuracy (95% Confidence Interval)		
			Lower Limit	Mean	Upper Limit	Lower Limit	Mean	Upper Limit	Lower Limit	Mean	Upper Limit
1	50	0.101	0.495	0.522	0.549	0.958	0.969	0.980	0.835	0.861	0.888
2	50	0.251	0.532	0.543	0.554	0.994	0.996	0.999	0.632	0.664	0.696
3	100	0.100	0.573	0.616	0.658	0.992	0.996	1.000	0.995	0.997	1.000
4	100	0.251	0.599	0.634	0.669	0.991	0.994	0.998	0.969	0.977	0.984
5	200	0.100	0.948	0.966	0.984	-	1.000	-	-	1.000	-
6	200	0.250	0.745	0.828	0.911	-	1.000	-	-	1.000	-
7	50	0.183	0.365	0.378	0.392	0.998	0.999	1.000	0.491	0.504	0.518
8	50	0.081	0.137	0.158	0.178	0.694	0.735	0.776	0.587	0.630	0.674
9	105	0.057	0.464	0.507	0.550	0.999	1.000	1.000	0.938	0.951	0.963
10	140	0.136	0.257	0.283	0.308	0.896	0.932	0.968	0.551	0.595	0.639
11	200	0.045	0.335	0.432	0.529	0.990	0.996	1.000	0.948	0.974	1.000
12	200	0.095	0.211	0.258	0.305	0.926	0.970	1.014	0.573	0.672	0.771
13	50	0.115	0.881	0.898	0.915	0.921	0.938	0.955	0.918	0.934	0.951
14	50	0.216	0.875	0.891	0.907	0.992	0.996	0.999	0.919	0.935	0.951
15	100	0.059	0.988	0.992	0.996	0.997	0.998	1.000	0.998	0.999	1.000
16	100	0.114	0.995	0.997	1.000	-	1.000	-	-	1.000	-
17	200	0.030	-	1.000	-	-	1.000	-	-	1.000	-
18	200	0.058	-	1.000	-	-	1.000	-	-	1.000	-

For 50% of the trials, the utility-score heuristic (UT) has a significantly higher average accuracy than the other two heuristics. For 15 of the 18 trials, NA had a significantly lower accuracy when compared to the other two heuristics. For five trials (3, 5, 6, 11, and 13) there is not enough evidence to support that either UT or MS has a

higher accuracy. The same is true for trials 16, 17, and 18, but for all three heuristics. These comparisons were made using hypothesis testing with the t-test statistic.

Table 17 shows the screening heuristics' average accuracies for each network type for the top 1, 5, 10, and 25 scenarios. These averages were calculated using all trials for each respective network type. To compensate for the differing number of runs per trial, 20 runs were randomly selected from each trial for the calculation of the average and CI. This ensured that each trial was equally represented in the calculations.

Table 17. Utility-Score Screening is Consistently More Accurate Across Different Network Types (Highlight Indicates Significantly More Accurate Heuristic for Network Type)

Network Type	Neighbor-Access Average Accuracy (95% Confidence Interval)				Utility Score Average Accuracy (95% Confidence Interval)				SNA Measure Average Accuracy (95% Confidence Interval)			
	Top 1	Top 5	Top 10	Top 25	Top 1	Top 5	Top 10	Top 25	Top 1	Top 5	Top 10	Top 25
ER	0.925 ± 0.048	0.857 ± 0.045	0.816 ± 0.043	0.692 ± 0.040	1.000	1.000	0.998 ± 0.002	0.990 ± 0.007	1.000	1.000	0.997 ± 0.003	0.916 ± 0.027
RC	0.467 ± 0.091	0.445 ± 0.063	0.401 ± 0.051	0.342 ± 0.035	0.960 ± 0.036	0.957 ± 0.032	0.943 ± 0.032	0.940 ± 0.025	0.825 ± 0.069	0.828 ± 0.048	0.798 ± 0.043	0.720 ± 0.040
BA	1.000	1.000	0.994 ± 0.005	0.970 ± 0.010	1.000	1.000	1.000	0.990 ± 0.006	1.000	1.000	1.000	0.981 ± 0.009

Reviewing the accuracy results from the perspective of network type, emphasizes the UT's consistency in accuracy across all network types. For both ER and RC networks, the UT has a significantly higher average accuracy than the other two heuristics. All three heuristics perform exceptionally well on BA networks with accuracies above 0.95.

Figure 19 consists of three boxplots, one for each network type. The boxplot shows the dispersion of run accuracies per trial (T#) and screening heuristic when examining the top 25 scenarios. Inconsistency is more prevalent in a trial when there is greater spread of accuracy values for the trial. These graphs provide insight into the consistency of the three screening heuristics in terms of accuracy. UT displays the

greatest consistency of the three heuristics across all network types and sizes. UT does show some inconsistency with the size-50 ER and BA networks. The greatest inconsistency displayed by UT is on Trial 8 and 10, both of which are RC networks of different size and parameters.

Both NA and MS, like UT, show the most consistency with BA networks, and like UT, do experience some inconsistency with smaller BA networks. NA is inconsistent for all ER and RC trials. MS is inconsistent for all RC trials and size-50 ER networks.

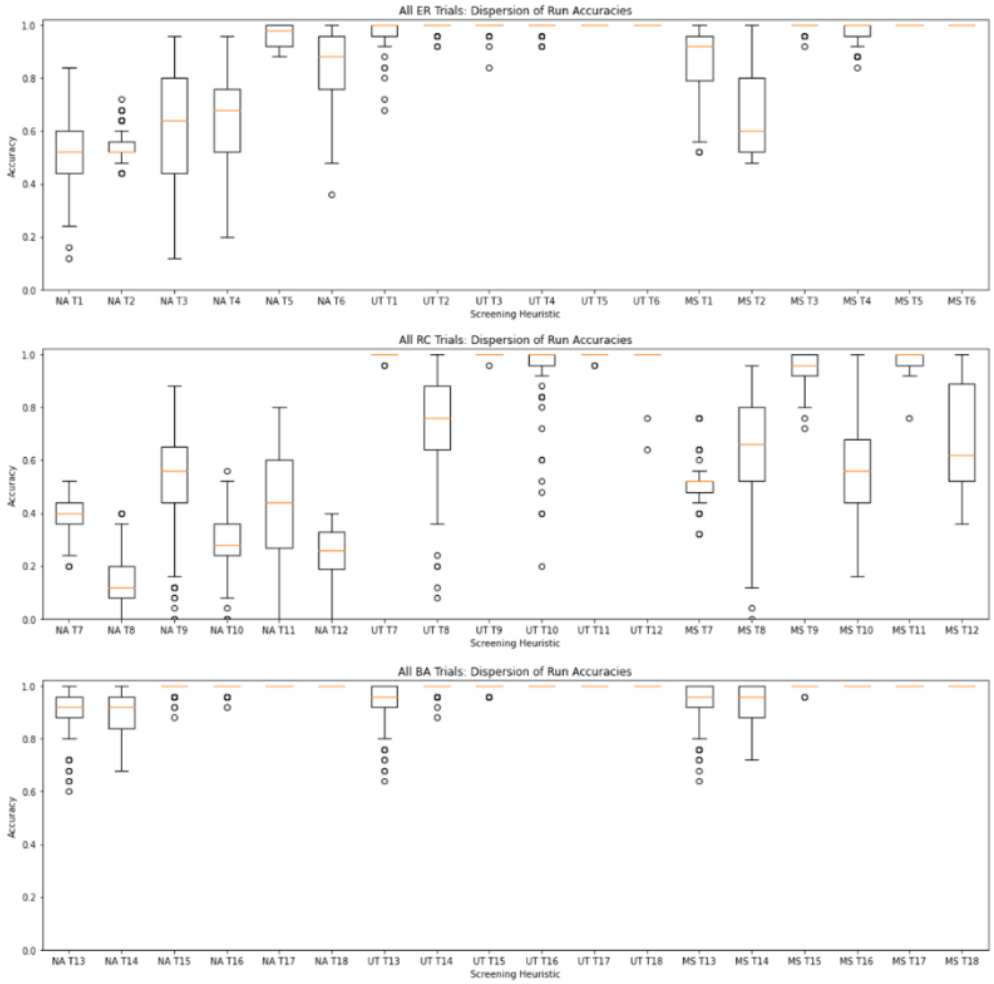

Figure 19. Utility-Score Screening Heuristic Maintains the Greatest Consistency Across All Trials & Network Types

4.5.2 Time Results

For each run, four processing times were collected, one for each screening heuristic and one for the "brute force" all-scenarios model. Table 18 shows the 95% confidence interval for the average processing time for each trial. Overall, a 99% reduction in processing time is achieved by the three heuristics when compared to the processing time for the all-scenarios model.

Table 18. Screening Heuristics Decrease Processing Time by 99% when Compared to All-Scenarios Model

Trial	All-Scenarios Model CPU Time (seconds)			NA CPU Time (seconds)			UT CPU Time (seconds)			MS CPU Time (seconds)		
	95% CI Lower Limit (LL)	Mean	95% CI Upper Limit (UL)	LL	Mean	UL	LL	Mean	UL	LL	Mean	UL
1	13.02	13.24	13.46	0.17	0.18	0.19	0.10	0.10	0.10	0.08	0.08	0.08
2	16.38	16.94	17.50	0.24	0.25	0.27	0.13	0.13	0.14	0.10	0.10	0.11
3	272.28	276.41	280.54	2.29	2.40	2.51	1.25	1.28	1.30	1.13	1.15	1.17
4	429.55	436.69	443.83	4.14	4.35	4.56	2.10	2.17	2.24	1.90	1.98	2.05
5	9707.74	9852.08	9996.43	55.97	63.55	71.13	40.31	41.61	42.91	39.20	39.77	40.34
6	16963.90	17536.44	18108.99	117.91	137.58	157.25	72.06	78.46	84.86	70.51	76.72	82.93
7	14.30	14.43	14.55	0.21	0.22	0.22	0.12	0.12	0.13	0.10	0.10	0.11
8	13.99	14.30	14.62	0.17	0.18	0.19	0.11	0.11	0.12	0.10	0.11	0.11
9	306.11	309.85	313.59	2.72	2.83	2.94	1.53	1.57	1.61	1.41	1.46	1.51
10	1726.29	1755.44	1784.59	14.22	14.98	15.75	10.68	11.73	12.79	8.05	8.54	9.04
11	6558.89	6661.84	6764.79	37.60	40.59	43.58	27.75	30.05	32.35	26.92	28.68	30.45
12	9225.80	9495.86	9765.92	58.54	69.35	80.15	54.83	63.33	71.83	45.98	50.80	55.63
13	12.91	13.03	13.15	0.16	0.16	0.17	0.10	0.10	0.10	0.08	0.08	0.08
14	17.22	17.53	17.84	0.22	0.23	0.23	0.13	0.13	0.14	0.11	0.11	0.11
15	224.95	227.15	229.35	1.63	1.64	1.66	1.02	1.03	1.04	0.92	0.93	0.95
16	285.46	289.31	293.16	2.04	2.07	2.10	1.30	1.31	1.33	1.17	1.19	1.21
17	5111.81	5175.49	5239.16	26.02	26.44	26.86	20.55	20.94	21.32	20.07	20.57	21.08
18	7516.34	7634.52	7752.71	37.75	38.69	39.64	30.63	31.25	31.86	29.73	30.25	30.77

The performance results show that UT provides the greatest consistency and average accuracy across all trials and network types, but there is no significant difference between UT's and MS's performance on the largest networks. With respect to time, all heuristics reduce the required processing time by 99%.

4.5.3 The Accuracy-Time Trade Space

Within the trade space of accuracy and time, more time does not necessarily increase accuracy. This is the result of how the screening heuristics operate. At a basic level, there are two reasons for reduced accuracy. First, if a top scenario or one of its targets is screened out by the screening mechanism the accuracy will be decreased. Any top ranked scenario that is screened out will never be evaluated by the heuristic and will never be included in the final list of scenarios, which reduces the accuracy. Second, if a top ranked scenario is positioned near the end of the scenario or target list, it is likely that the search heuristic will halt prior to the scenario being evaluated. If it is not evaluated by the heuristic then it is not included in the final ranked list of scenarios, again reducing accuracy.

To counter the above effects, two actions can be taken. First, μ or β can be increased allowing for more scenarios to be kept by the screening mechanisms. This alone will not increase accuracy. Increasing the stagnation criteria, the second action, will be required to ensure that the new scenarios in the scenario list are evaluated by the heuristic. Both of these actions will increase the processing time of the screening heuristics. If, after the increases in the hyperparameters, top scenarios are still screened out by the screening mechanism the increased processing time will not result in increased accuracy. Analyst judgement will be required to determine if the additional processing time is worth the resulting increase in accuracy.

4.6 Analysis of Heuristic Performance

As stated in Chapter I, the purpose of this thesis is to provide a method to reduce the computational time required to produce a list of the most advantageous insertion

scenarios within a covert network, while maintaining an adequate overall level of accuracy. In the above results section, the three screening heuristics presented drastically reduce the required computational time, but only one heuristic, UT, provides a consistent and adequate level of accuracy across varying network types. The following analysis focuses on network characteristics that affect network accuracy. Furthermore, the above time results will be used to estimate the expected computational time required to evaluate larger networks than those used within this research.

4.6.1 Network Characteristics and Heuristic Accuracy

Understanding the operations of the screening heuristics provides awareness of two fundamental reasons why a heuristic might miss a top scenario, and thus reduce its accuracy. The first reason, which applies to UT and MS, is that a key target is screened out by the screening function, and therefore not included in the target list that is passed to the search heuristic. Any scenario that includes that target will now be outside the solution space, will not be discovered by the heuristic, and will never be evaluated. The same applies to NA, where if a scenario does not make the lists that are passed to the heuristic, the scenario will never be discovered and evaluated by the heuristic. The second reason for a screening heuristic missing a top scenario is that the scenario is situated too far down the target or scenario list that the heuristic meets the stagnation criteria and halts before the scenario is up for evaluation.

Despite understanding the above reasons for reduced accuracy, it is important to understand which network characteristics affect the accuracy of the heuristics. The focus of the following analysis is to determine network factors that have the greatest effect on determining a heuristics' accuracy.

Upon inspection of the results, it can be noted that both network structure and size contribute to the accuracy of all three screening heuristics, as all three have a noticeable decrease in accuracy when evaluating RC networks and networks with 50 nodes. In comparison, all three heuristics achieve high accuracy when evaluating BA networks. These insights suggest that certain key measures of network structure, such as degree distribution, modularity, and network size, can provide a better understanding of the factors that contribute to heuristic accuracy.

Figure 20 is the correlation matrix that explores the relationships between the accuracies of the three screening heuristics and key network measures. This matrix was developed using information from all runs across all trials.

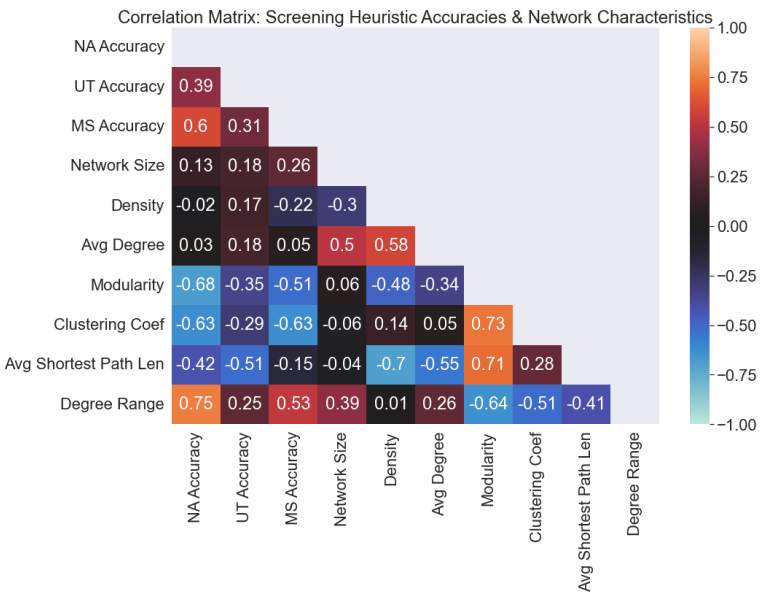

Figure 20. There are Strong & Moderate Correlations (Positive & Negative) Between Network Characteristics & Heuristic Accuracy

The highest correlation observed between a heuristic's accuracy and a network measure is a strong, positive correlation between NA accuracy and degree range, the

range between the maximum degree and minimum degree in the network. Modularity and clustering coefficient have a negative, moderate correlation with one or more of the heuristic accuracies. These results highlight that the structure of the network being analyzed is a key factor in determining the accuracy of the screening heuristic.

To explore the correlation between accuracy and degree range, graphs are presented in Figure 21 depicting the relationship between the two. Both NA and MS are less accurate on networks when there is a smaller range between the maximum and minimum degree values. Degree range has less of an effect on UT accuracies.

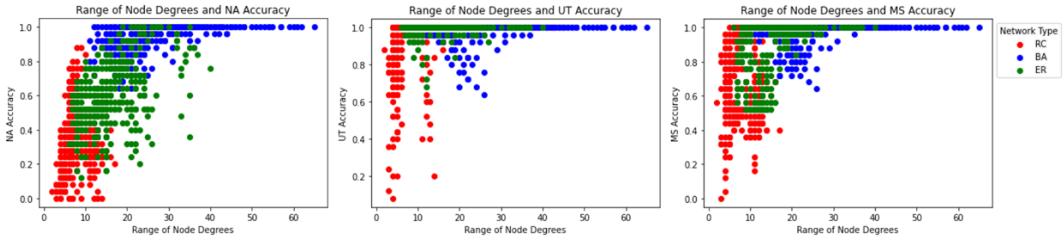

Figure 21. Degree Range has a Strong Correlation with NA Accuracy & Moderate Correlation with MS Accuracy

The relationship between degree range and accuracy is logical as the screening criteria for both NA and MS heavily rely on node degree. MS is less affected by degree range due to eigenvector centrality being a weighted sum of connections compared to the strict sum of connections in degree centrality. Neighbor-access scores are a mere extension of the targets' degrees.

Figure 22 shows the histograms of node degree and neighbor-access scores for two runs to highlight the effect of degree range on NA accuracy. The left group of graphs is from Trial 8, Run 52 (RC, $n = 50, p = 0.25, k = 5, l = 10$) which has an accuracy of zero when evaluated by NA. The right graphs are from Trial 14, Run 24

(BA, $n = 50, m = 6$) which has an accuracy of 1.00 when evaluated by NA. The BA graph has a larger range of degree values, which results in a wider distribution of neighbor-access scores. The opposite is seen with the RC graph: a smaller range of degree values and a narrower distribution of neighbor-access scores.

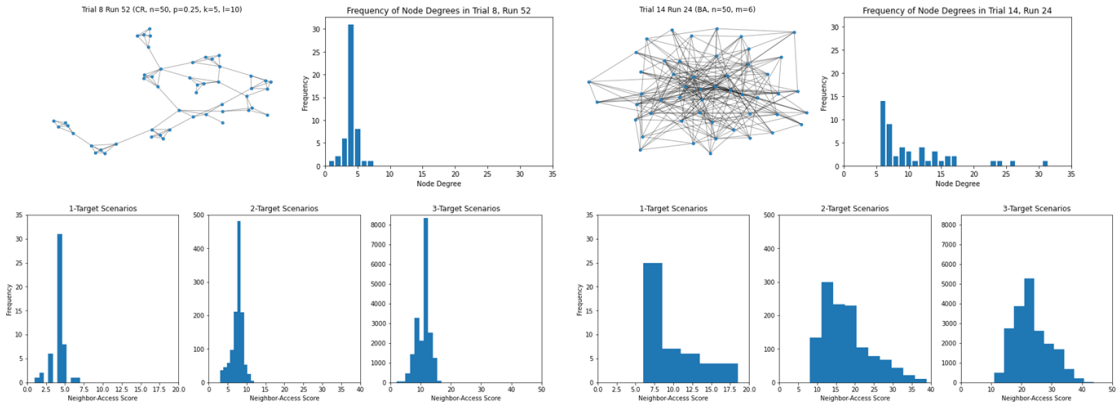

Figure 22. A Smaller Range of Degree Values Results in a Smaller Range of Neighbor-Access Scores, Resulting in Lower NA Accuracy

As discussed in Chapter III, NA screening calculates the scores for all scenarios in each scenario type (one-, two-, and three-target). Then, within each scenario type, the scenarios are sorted from highest to lowest by neighbor-access score and the top β scenarios are selected. With a narrow distribution of scores, the top β scenarios could all have similar scores, providing little differentiation. Furthermore, randomness is introduced depending on where the β falls, separating the scenarios that are kept and those that are discarded. For instance, after sorting scenarios by score, if the 25th through 75th scenarios all have the same score and $\beta = 50$, then half of those scenarios will be discarded, even though they meet the same criteria as those that are kept. This results in scenario selection based on how the sorting function operates and not by scenario

characteristics, therefore increasing randomness in the screening process. This is a weakness in NA that will need to be addressed in future research.

Network modularity and clustering coefficient were both moderately correlated with NA and MS accuracy. Unlike degree range, these correlations are negative. Clustering or the presence of groups within a network can affect heuristic accuracy similarly to degree range. Members of the clusters or groups tend to be highly connected to each other, which results in nodes with similar measures and characteristics. This in turn makes it more difficult for the screening process to differentiate between the nodes, increasing the likelihood that a key target is excluded from the target list passed to the heuristic.

Like degree range, neither modularity nor clustering is the single factor that indicates a heuristics performance, but one of many. This is evident when looking specifically at the ER trials (green) in the above graphs. The ER networks have some of the lowest modularity and clustering coefficients of all runs, but NA and MS still struggle to accurately evaluate these networks. Therefore, there are other factors that contribute to determining a heuristic's accuracy.

The final correlated network characteristic is a network's average shortest path length (ASPL), which has a moderate negative correlation with UT accuracy. This correlation is due to UT having a higher average accuracy on BA networks, which have lower ASPLs when compared to ER and RC networks.

To further explore the relationships between degree range, modularity, clustering, and ASPL, linear regression was performed to determine which of the network characteristics are good predictors for accuracy. Additional network measures were

included in this analysis: network type, density, and number of nodes. Due to multicollinearity, only network density, modularity, and degree range were included in the final regression model. When predicting NA, UT, and MS accuracy, all three were sufficient predictors resulting in an overall R^2 of 0.91, 0.88, and 0.86 respectively.

The overarching insight gained from the relationship between heuristic accuracy and the specific network measures is that network type plays an important role in determining the accuracy of the heuristics. As discussed in Chapter II, BA and RC networks have specific network characteristics. BA networks are scale-free networks whose degree distribution follows a power law, resulting in a higher probability of highly connected hub-nodes and a wider degree range. In addition, the ASPL of BA networks tend to be small because of the higher number of hub-nodes. The modularity and clustering coefficients of BA networks are also lower. RC networks on the other hand are characterized by their high modularity and clustering coefficient. In addition, RC networks can have high ASPL. ER network generation is more random than the other two types, and therefore ER networks can potentially take on characteristics of both BA and RC networks. In Figure 21 there is a discernable difference between the BA and RC runs, and the respective accuracy of the heuristics when evaluating each network type.

Ultimately, certain network characteristics result in targets of similar measures and characteristics making it difficult for the screening mechanisms to differentiate between the individual targets. The differences in the average accuracies of the three heuristics can be explained by the amount of information captured by their respective screening criteria. As mentioned before, NA is a modification of the total degree of the targets within a scenario, resulting in the least amount of information captured about

those targets and scenario. Measure screening, which uses eigenvector centrality, captures additional information since eigenvector centrality is a weighted degree of the targets, taking into consideration the quality of each connection. Finally, UT screening captures the most information about a target or scenario as it has elements of eigenvector, betweenness, and closeness centralities. This additional target information allows UT to differentiate between targets more easily during the screening process.

4.6.2 Estimating Performance on Larger Networks

Using the time results presented in Section 4.5.2, the expected computational time required for evaluation of larger networks were estimated. These estimates are based on a μ and β set to 25% of network size and stagnation criteria set to 25% of the possible scenarios per μ and β. The processing times roughly follow a power law distribution. Figure 23 shows the expected time required to evaluate larger networks using the all-scenarios model. For a network of size 500, the estimated time is 135 hours. The time increases to just under 3,000 hours when evaluating a 1,000-node network.

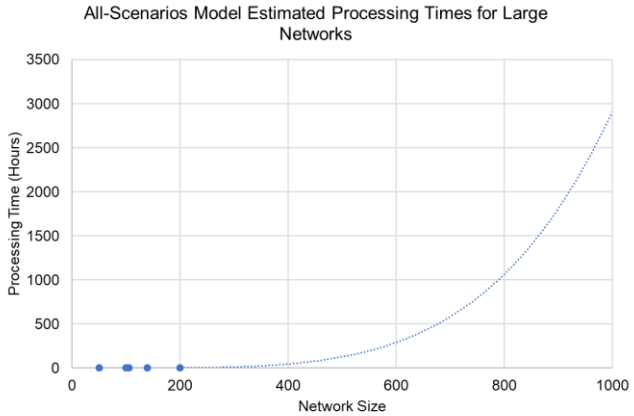

Figure 23. For a 500-Node and 1,000-Node Network the All-Scenario's Model Required Computational Time is 135 Hours and 2,985 Hours, Respectively

Figure 24 shows the estimated times for the three heuristics. All three heuristics are estimated to evaluate a 1,000-node network between 5.5 and 7.5 hours.

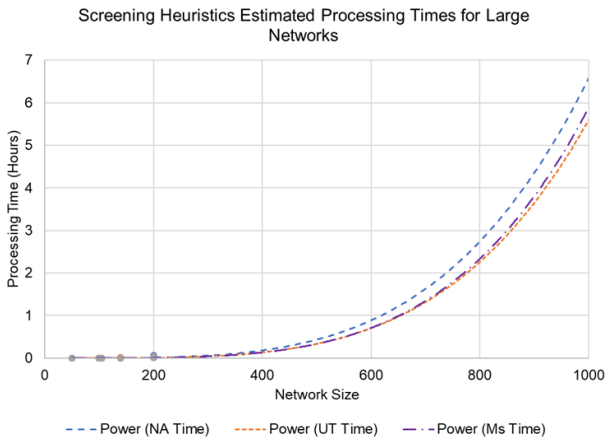

Figure 24. All 3 Heuristics are Estimated to Evaluate a 1,000-Node Network in Less Than 8 Hours

These time estimates are for a Dell Inspiron 5515 Laptop with 16 GB of RAM and an AMD Ryzen 7 5700U (1.8GHz) processor. Applying increased processing power and leveraging parallel processing can greatly reduce these time estimates.

Estimating the accuracy of the three heuristics is a little more difficult than estimating time. There was an increase in accuracy seen with all three heuristics when evaluating the larger networks during testing, which could signal that all three heuristics will achieve a high level of accuracy on larger networks. More testing is required to confirm this. It is expected that NA will still perform poorly on larger RC networks, based on its performance during testing.

4.7 Chapter Summary

This chapter evaluated the effects of changes to the utility function's cost factor and weights on the ranking of insertion scenarios. Then the hyperparameters were

evaluated to understand their effects on heuristic accuracy and processing time. These findings were included into final performance testing of the three heuristics. The results were followed by analysis of network characteristics and their effects on accuracy. Finally, heuristic performance on networks larger than those used during test was estimated.

Overall heuristic performance varied by heuristic and network types. UT was the most consistent and achieved the highest average accuracy for two of the three network types. MS achieved the fast average time for a majority of the trials. All three heuristics showed that they performed better on the larger networks used during testing.

V. Conclusions and Recommendations

5.1 Chapter Overview

This chapter discusses the conclusions and significance of the research and highlights future research topics within the realm of node insertion.

5.2 Conclusions of Research

This research presented three screening heuristics to evaluate node insertion scenarios within a covert network. The three screening heuristics each employed separate screening criteria − neighbor-access score, utility score, and SNA measure − to identify the most promising targets or scenarios to reduce the solution space prior to the application of a search heuristic. The heuristics utilized a quantitative utility scoring method that consisted of a weighted benefit and risk score, each calculated using common SNA measures.

The three heuristics were tested on three types of randomly generated graphs, Erdős–Rényi, Relaxed Caveman, and Barabási-Albert. All three heuristics were successful in reducing the computational time by 99% when compared to the "brute-force" all-scenarios model. Only the utility-score heuristic was able to maintain at least a 0.90 average accuracy across all network types when comparing the top 25 scenarios. The neighbor-access heuristic was the least accurate and most inconsistent of the three heuristics, achieving an average accuracy of 0.342 ± 0.034 on RC networks, 0.692 ± 0.010 on ER networks, and 0.970 ± 0.010 on BA networks. The SNA measure heuristic performed well on both ER and BA networks, achieving an average accuracy of 0.916 ±

0.027 and 0.981 ± 0.009, respectively, but struggled with RC networks where it only achieved an average accuracy of 0.720 ± 0.040.

Multiple factors affect the performance of the three heuristics. Characteristics of a network's structure, specifically modularity, clustering, degree range, and average shortest path length, were shown to affect the accuracy of the heuristics by making it more difficult to differentiate between promising targets or scenarios during the screening step. In addition, parameters within the utility function and the hyperparameters of the heuristics can also affect the performance of the heuristics and will require tuning to obtain the desired balance of accuracy and processing time.

5.3 Significance of Research

This research provides intelligence and law enforcement officials with a computationally inexpensive method to identify advantageous node insertion scenarios within a given covert network. This method is not intended to be the deciding factor in determining which insertion scenario to pursue but should be used in the initial steps of course of action development. The identified advantageous scenarios will require further analysis by intelligence and law enforcement SMEs to ensure other critical information and network vulnerabilities are incorporated in the final analysis of the insertion scenarios. The presented methods are developed to provide the intelligence and law enforcement analysts with a quick and quantitative method to down select the number of potential node insertion scenarios, reducing time requirements for follow-on analyses.

5.4 Recommendations for Future Research

This thesis serves as an initial foundation for research of node insertion into covert networks for the purpose of disruption or information gathering. There are many

directions that can be taken for future research into this subject area, and many include methods that were discussed in Chapter II for network disruption through node deletion.

Directly related to this research is operationalizing the heuristics into a tool that can be easily operated by an intelligence or a law enforcement analyst. This would require the development of a graphical user interface that would allow the analyst to build a model of the covert network, establish parameters and hyperparameters, and then apply the heuristic evaluation method to output advantageous insertion scenarios.

This research is limited in its approach as it only utilizes static, undirected, unweighted networks in its development and analysis of the heuristics. Important network aspects were not included in this analysis that may have dramatic effects on the selection of advantageous insertion scenarios. These include geographical location of network members, strength and type of relationships between members, and attributes of each member. Future research can build upon this study by incorporating these aspects into the network models, and then modifying the presented heuristics to accommodate for the additional information.

A similar approach to that used by Geffre et al. [37] and Carley et al. [35], models the network by utilizing a meta-matrix approach that captures the member, knowledge, resources, and task aspects of the network. This approach determines where the cognitive load and expertise lie within a network. This information can be leveraged for determining potential node insertion scenarios.

Future research can utilize a multi-layered networking method to incorporate additional information about the covert network into the network model. This method splits the network into various layers to capture the different types of connections

between members. These layers can include cyber, financial, social/familial, and operational. Future research can study how to perform insertion into one layer in order to achieve a desired effect in another layer of the network.

A potential area of research could be the study of where to insert an agent when specifically targeting certain members of a network. Having the agent develop direct relationships with these targets would be extremely risk. Future research can study where in the network to insert the agent and then which existing relationships in the network the agent can leverage to gain access to the targets.

After selecting the most advantageous node insertion scenarios, dynamic modeling of the network and the use of game theory can be used to simulate each node insertion scenario, and the network's response to the insertion. This simulation can help to further evaluate the risks and benefits of each scenario as well as test potential insertion strategies.

5.5 Summary

This research presents three screening heuristics to evaluate potential node insertion scenarios, with one, utility-score screening, being the highest performing and showing the most potential for evaluating larger networks than those used during this study. Due to the lack of access to realistic data of covert networks and a method to evaluate if the utility function properly identifies the highest scored insertion scenario, the proposed methods of this research are only theoretical. Future research can further develop various applications within the area of covert network node insertion, providing better methods for intelligence and law enforcement analyst to leverage when determining future agent insertion operations.

Appendix A. Python Code

In [ ]:
```
import networkx as nx
import pandas as pd
import numpy as np
import itertools as it
from networkx.algorithms import approximation as app
import statistics
import time
import igraph as ig
from igraph import *
from tqdm.notebook import trange, tqdm, tqdm_notebook
import random as rd
```

All Scenarios Algorithm and sub-functions

In [ ]:
```
def pairs(nodes):
    '''
    Creates a list of all possible two-target scenarios based upon a given list.  Maintains node order within list.

    Input:
    nodes: a list of nodes

    Output:
    pairs: a list of tuples, where each tuple represents a possible two-target scenario
    '''
    pairs = []

    for item in nodes:
        n = nodes.index(item)
        for x in range(n+1, len(nodes)):
            pairs.append((item, nodes[x]))
    return (pairs);
```

In [ ]:
```
def triplets(nodes):
    '''
    Creates a list of all possible three-target scenarios based upon a given list.  Maintains node order within list.

    Input:
    nodes: a list of nodes

    Output:
    triplets: a list of tuples, where each tuple represents a possible three-target scenario
    '''
    triplets = []

    for item in nodes:
        n = nodes.index(item)
        for x in range(n+1, len(nodes)-1):
            for y in range(x+1, len(nodes)):
                triplets.append((item, nodes[x], nodes[y]))
    return (triplets);
```

In [ ]:
```
def node_lists(G):
    '''
    Creates three lists: all nodes in the graph, all possible two-target scenarios, all possible three-target
    scenarios in the graph

    Input:
    G: a networkx graph

    Ouput:
    Nodes: (list of int) all nodes in the graph
    all_pairs: (list of tuples) all possible two-target scenarios
    all_triplets: (list of tuples) all possible three-target scenarios
    '''
    #Create list of nodes in graph
    Nodes = sorted(nx.nodes(G))

    all_pairs = pairs(Nodes)
    all_triplets = triplets(Nodes)

    return Nodes, all_pairs, all_triplets
```

In [ ]:

```
def one_relationship(G, Nodes, data, cost, weights, deg_assort_H, trial, run, n, algorithm):
    '''
    Calculates the utility score for each node in Nodes and outputs the score information to a dataframe

    Input:
    G: a networkx graph
    Nodes: (list of int): a list of all nodes in the graph
    data: dataframe with specific headers
    deg_assort_H (int): degree assortativity of G without agent in network
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    n: (int) number of nodes in the graph
    run: (int) run number

    Output:
    data: a dataframe with utility score information
    '''
    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * len(Nodes)  # empty list; will become list of lists, then finally dataframe

    #for each node in the list Nodes
    for i, target1 in enumerate(Nodes):

        #create edge between the agent and the target
        G.add_edge(n, target1)
        H = ig.Graph.from_networkx(G)

        #calculate the network's degree assortativity with new edges between agent and targets
        deg_assort_w_agent = H.assortativity_degree(directed=False)

        #calculate the betweenness centrality for all nodes
        Betw = H.betweenness(vertices = [target1], directed = False, cutoff = None, weights=None)
        Betw_Cent = []
        for c in range(len(Betw)):
            new = Betw[c]*(2/((n-1)*(n-2)))
            Betw_Cent.append(new)

        #calculate the eigenvector centrality for all nodes
        Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
        #Eig_Cent = normalize(Eig_Cent)

        #calculate the closeness centrality for all nodes
        Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
        #Close_Cent = normalize(Close_Cent)

        #calculate the benefit score using above measures and respective weights out weights list
        benefit = weights[0] * Betw_Cent[0] + weights[1] * Eig_Cent[target1]

        #calculate the risk score using above measures and respective weights out weights list
        risk = weights[2] * Close_Cent[0] + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 1*cost

        #calculate the utility score using benefit and risk and respective weights
        utility = weights[4] * benefit - weights[5] * risk

        #write data to dataframe data
        tmp_data_list[i] = [trial, run, algorithm, tuple((target1, 'NA', 'NA')), deg_assort_H, deg_assort_w_agent,
                            Close_Cent[0], Eig_Cent[target1], Betw_Cent[0],
                            'NA', 'NA',
                            'NA', 'NA',
                            benefit, risk, utility]

        #remove edge from between agent and target
        G.remove_edge(n, target1)

    # convert list of lists to a dataframe
    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
    #    remove it if you don't care what the 'row number' is in your output
    tmp_dataframe.index += 1

    # combine with original dataframe before returning
    data = pd.concat([data, tmp_dataframe], ignore_index=False)

    return (data);
```

In [ ]:

```
def two_relationship(G, pairs, data, cost, weights, deg_assort_H, trial, run, n, algorithm):
    '''
    Calculates the utility score for each scenario in pairs and outputs the score information to a dataframe

    Input:
    G: a networkx graph
    pairs: (list of tuples): a list of all possible two-target scenarios in graph
    data: dataframe with specific headers
    deg_assort_H (int): degree assortativity of G without agent in network
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    n: (int) number of nodes in the graph
    run: (int) run number

    Output:
    data: a dataframe with utility score information
    '''
    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * len(pairs)  # empty list; will become list of lists, then finally dataframe

    #for each pair in the list pairs
    for i, (target1, target2) in enumerate(pairs):

        #create edges between the agent and the targets
        G.add_edge(n, target1)
        G.add_edge(n, target2)
        H = ig.Graph.from_networkx(G)

        #calculate the network's degree assortativity with new edges between agent and targets
        deg_assort_w_agent = H.assortativity_degree(directed=False)

        #calculate the betweenness centrality for all nodes
        Betw = H.betweenness(vertices = [target1, target2], directed = False, cutoff = None, weights=None)
        Betw_Cent = []
        for c in range(len(Betw)):
            new = Betw[c]*(2/((n-1)*(n-2)))
            Betw_Cent.append(new)

        #calculate the eigenvector centrality for all nodes
        Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
        #Eig_Cent = normalize(Eig_Cent)

        #calculate the closeness centrality for all nodes
        Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
        #Close_Cent = normalize(Close_Cent)

        #calculate the benefit score using above measures and respective weights out weights list
        benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2])

        #calculate the risk score using above measures and respective weights out weights list
        risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 2*cost

        #calculate the utility score using benefit and risk and respective weights
        utility = weights[4] * benefit - weights[5] * risk

        #write data to dataframe data
        tmp_data_list[i] = [trial, run, algorithm, tuple((target1, target2, 'NA')), deg_assort_H, deg_assort_w_agent,
                            Close_Cent[0], Eig_Cent[target1], Betw_Cent[0],
                            Eig_Cent[target2], Betw_Cent[1],
                            'NA', 'NA',
                            benefit, risk, utility]

        #remove edges from between agent and targets
        G.remove_edge(n, target1)
        G.remove_edge(n, target2)

    # convert list of lists to a dataframe
    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
    #    remove it if you don't care what the 'row number' is in your output
    tmp_dataframe.index += 1

    # combine with original dataframe before returning
    data = pd.concat([data, tmp_dataframe], ignore_index=False)

    return (data);
```

In [ ]:

```
def three_relationship(G, triplets, data, cost, weights, deg_assort_H, trial, run, n, algorithm):
    '''
    Calculates the utility score for each scenario in triplets and outputs the score information to a dataframe

    Input:
    G: a networkx graph
    triplets: (list of tuples): a list of all possible two-target scenarios in graph
    data: dataframe with specific headers
    deg_assort_H (int): degree assortativity of G without agent in network
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    n: (int) number of nodes in the graph
    run: (int) run number

    Output:
    data: a dataframe with utility score information
    '''
    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * len(triplets)  # empty list; will become list of lists, then finally dataframe

    #for each triplet in the list triplets
    for i, (target1, target2, target3) in enumerate(triplets):

        #create edges between the agent and the targets
        G.add_edge(n, target1)
        G.add_edge(n, target2)
        G.add_edge(n, target3)
        H = ig.Graph.from_networkx(G)

        #calculate the network's degree assortativity with new edges between agent and targets
        deg_assort_w_agent = H.assortativity_degree(directed=False)

        #calculate the betweenness centrality for all nodes
        Betw = H.betweenness(vertices = [target1, target2, target3], directed = False, cutoff = None, weights=None)
        Betw_Cent = []
        for c in range(len(Betw)):
            new = Betw[c]*(2/((n-1)*(n-2)))
            Betw_Cent.append(new)

        #calculate the eigenvector centrality for all nodes
        Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
        #Eig_Cent = normalize(Eig_Cent)

        #calculate the closeness centrality for all nodes
        Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
        #Close_Cent = normalize(Close_Cent)

        #calculate the benefit score using above measures and respective weights out weights list
        benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]+ Betw_Cent[2]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2]+ Eig_Cent[target3])

        #calculate the risk score using above measures and respective weights out weights list
        risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 3*cost

        #calculate the utility score using benefit and risk and respective weights
        utility = weights[4] * benefit - weights[5] * risk

        #write data to dataframe data
        tmp_data_list[i] = [trial, run, algorithm, tuple((target1, target2, target3)), deg_assort_H, deg_assort_w_agent,
                            Close_Cent[0], Eig_Cent[target1], Betw_Cent[0],
                            Eig_Cent[target2], Betw_Cent[1],
                            Eig_Cent[target3], Betw_Cent[2],
                            benefit, risk, utility]

        #remove edges from between agent and targets
        G.remove_edge(n, target1)
        G.remove_edge(n, target2)
        G.remove_edge(n, target3)

    # convert list of lists to a dataframe
    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
    #    remove it if you don't care what the 'row number' is in your output
    tmp_dataframe.index += 1

    # combine with original dataframe before returning
    data = pd.concat([data, tmp_dataframe], ignore_index=False)

    return (data);
```

[ ]:
```
def all_scenarios(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, trial, run):
    '''
    Calculates the utility score for all possible one-, two-, and three- target scenarios in graph

    Input:
    G: a networkx graph
    Nodes: (list of int) list of all nodes in graph
    all_pairs: (list of tuples): a list of all possible two-target scenarios in graph
    all_triplets: (list of tuples): a list of all possible three-target scenarios in graph
    deg_assort_H (int): degree assortativity of G without agent in network
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    n: (int) number of nodes in the graph
    trial: (int) trial number
    run: (int) run number

    Output:
    data: a dataframe with utility score information
    '''
    data4 = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                    'CC Agent','EC T1', 'BC T1',
                                    'EC T2', 'BC T2',
                                    'EC T3', 'BC T3',
                                    'Benefit', 'Risk', 'Utility'])
    algorithm = "All"

    start = time.perf_counter()

    #Add agent to network
    G.add_node(n)

    #Calculate utility score for all single-target scenarios
    data4 = one_relationship(G, Nodes, data4, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Calculate utility score for all two-target scenarios
    data4 = two_relationship(G, all_pairs, data4, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Calculate utility score for all three-target scenarios
    data4 = three_relationship(G, all_triplets, data4, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Remove agent from network
    G.remove_node(n)

    end = time.perf_counter()

    total_time = end-start

    return data4, total_time
```

Utility Score Screening Heuristic

[ ]:
```
def best_nodes_by_utility(data, a):
    '''
    Creates a list of the top 'a' nodes with the highest single utility scores found using one_relationship().

    Input:
    data: dataframe output from one_relationship()
    a: (int) top number of nodes to keep

    Output:
    best_nodes_utility: a list of the top 'a' nodes with the highest single utility scores
    '''
    data_sorted = data.sort_values(by="Utility", ascending = False)
    top_scenarios = data_sorted['Scenario'].to_list()
    top_nodes = []
    for item in top_scenarios:
        top_nodes.append(item[0])
    best_nodes_utility = top_nodes[:a]

    return best_nodes_utility;
```

```
def get_two_relationship_data(G, data, deg_assort_H, top_nodes, stale, iterations, cost, weights, trial, run, algorithm):
    '''
    Performs a local search on the list top_nodes to determine the pair of nodes with the highest utility score.
    The search iteraties, starting with the first node in the list, pairing it with all other nodes in the list
    calculating each pair's utility score. The search then moves to the second node, pairing it with all other
    nodes in the list, calculating the utility scores of each of those pairs. This process continues until either
    the desired number of iterations is achieved or a stagnation criteria is met.

    Input:
    G: a networkx graph
    data (dataframe): a dataframe with specific headers
    deg_assort_H (int): degree assortativity of G without agent in network
    top_nodes (list of int): list of the top 'a' nodes
    stale (list of int): stagnation criteria
    iterations (list of int): the number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    trial: (int) trial number
    run: (int) run number
    algorithm: (str) algorithm being performed

    Outputs;
    data: dataframe with utility score information
    '''
    best = -5 # best utility score set to arbitrarily low number
    t = 0  # iteration counter
    s = 0  # stagnation counter
    i = 0

    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * iterations[0]  # empty list; will become list of lists, then finally dataframe

    stop = False
    while stop is False:
        for target1 in top_nodes:
            q = top_nodes.index(target1)
            for target2 in range(q+1, len(top_nodes)):
                target2 = top_nodes[target2]

                #create edges between the agent and the targets
                G.add_edge(n, target1)
                G.add_edge(n, target2)
                H = ig.Graph.from_networkx(G)

                #calculate the network's degree assortativity with new edges between agent and targets
                deg_assort_w_agent = H.assortativity_degree(directed=False)

                #calculate the betweenness centrality for all nodes
                Betw = H.betweenness(vertices = [target1, target2], directed = False, cutoff = None, weights=None)
                Betw_Cent = []
                for c in range(len(Betw)):
                    new = Betw[c]*(2/((n-1)*(n-2)))
                    Betw_Cent.append(new)

                #calculate the eigenvector centrality for all nodes
                Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
                #Eig_Cent = normalize(Eig_Cent)

                #calculate the closeness centrality for all nodes
                Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
                #Close_Cent = normalize(Close_Cent)

                #calculate the benefit score using above measures and respective weights out weights list
                benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2])

                #calculate the risk score using above measures and respective weights out weights list
                risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 2*cost

                #calculate the utility score using benefit and risk and respective weights
                utility = weights[4] * benefit - weights[5] * risk

                #write data to dataframe data
                if target1 > target2:
                    spot1 = target2
                    spot2 = target1
                else:
                    spot1 = target1
                    spot2 = target2
```

```
            tmp_data_list[i] = [trial, run, algorithm, tuple((spot1, spot2, 'NA')), deg_assort_H, deg_assort_w_agent,
                                Close_Cent[0],Eig_Cent[target1], Betw_Cent[0],
                                Eig_Cent[target2], Betw_Cent[1],
                                'NA', 'NA',
                                benefit, risk, utility]

            #remove edges between agent and targets
            G.remove_edge(n, target1)
            G.remove_edge(n, target2)



            #update iteration counter
            t += 1
            i += 1

             #if the new utility score is greater than the current best, make it the best, reset stagnation counter
            if utility > best:
                best = utility
                s = 0
            else: #if the new utility score is less than the current best, update the stagnation counter
                s += 1
                if s == stale[0]: #if stagnation counter is greater than stagnation criteria, stop search
                    stop = True

                    # convert list of lists to a dataframe
                    tmp_data_list = tmp_data_list[0:s]
                    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                    #    remove it if you don't care what the 'row number' is in your output
                    tmp_dataframe.index += 1

                    # combine with original dataframe before returning
                    data = pd.concat([data, tmp_dataframe], ignore_index=False)

                    print(s, t, 'too stale')
                    return data

            if t == iterations[0]: #if iteration counter is greater than max iterations, stop search
                stop = True

                # convert list of lists to a dataframe
                tmp_data_list = tmp_data_list[0:t]
                tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                #    remove it if you don't care what the 'row number' is in your output
                tmp_dataframe.index += 1

                # combine with original dataframe before returning
                data = pd.concat([data, tmp_dataframe], ignore_index=False)

                print(s, t, 'iteration limit')
                return data

    print(s, t, 'end of func')
```

```
def get_three_relationship_data(G, data, deg_assort_H, top_nodes, stale, iterations, cost, weights, trial, run, algorithm):
    '''
    Performs a local search on the list top_nodes to determine the triplet of nodes with the highest utility score.
    The search iteraties, starting with the first node in the list, creating a triplet with it and the next two nodes in
    in the list.  This continues iterating across all possible triplets starting with the first node in top_nodes. At
    each iteration it calculates each triplet's utility score. The search then moves to the second node, doing the same
    as with the first node. This process continues until either the desired number of iterations is achieved, a
    stagnation criteria is met, or all node triplets are exhausted.

    Input:
    G: a networkx graph
    data (dataframe): a dataframe with specific headers
    deg_assort_H (int): degree assortativity of G without agent in network
    top_nodes (list of int): list of the top 'a' nodes
    stale (list of int): stagnation criteria
    iterations (list of int): the number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    trial: (int) trial number
    run: (int) run number
    algorithm: (str) algorithm being performed

    Outputs;
    data: dataframe with utility score information
    '''
    best = -5 # best utility score set to arbitrarily low number
    t = 0  # iteration counter
    s = 0  # stagnation counter
    i = 0
```

```
    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * iterations[1]  # empty list; will become list of lists, then finally dataframe

    stop = False
    while stop is False:
        for target1 in top_nodes:
            q = top_nodes.index(target1)
            for x in range(q+1, len(top_nodes)-1):
                target2 = top_nodes[x]
                for y in range(x+1, len(top_nodes)):
                    target3 = top_nodes[y]

                    #add edges between agent and targets
                    G.add_edge(n, target1)
                    G.add_edge(n, target2)
                    G.add_edge(n, target3)
                    H = ig.Graph.from_networkx(G)

                    #calculate the network's degree assortativity with new edges between agent and targets
                    deg_assort_w_agent = H.assortativity_degree(directed=False)

                    #calculate the betweenness centrality for all nodes
                    Betw = H.betweenness(vertices = [target1, target2, target3], directed = False, cutoff = None, weights=None)
                    Betw_Cent = []
                    for c in range(len(Betw)):
                        new = Betw[c]*(2/((n-1)*(n-2)))
                        Betw_Cent.append(new)

                    #calculate the eigenvector centrality for all nodes
                    Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
                    #Eig_Cent = normalize(Eig_Cent)

                    #calculate the closeness centrality for all nodes
                    Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
                    #Close_Cent = normalize(Close_Cent)

                    #calculate the benefit score using above measures and respective weights out weights list
                    benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]+ Betw_Cent[2]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2]+ Eig_Cent[target3])

                    #calculate the risk score using above measures and respective weights out weights list
                    risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 3*cost

                    #calculate the utility score using benefit and risk and respective weights
                    utility = weights[4] * benefit - weights[5] * risk

                    #write data to dataframe data
                    tmp_data_list[i] = [trial, run, algorithm, tuple(sorted((target1, target2, target3))), deg_assort_H, deg_assort_w_agent,
                                        Close_Cent[0], Eig_Cent[target1], Betw_Cent[0],
                                        Eig_Cent[target2], Betw_Cent[1],
                                        Eig_Cent[target3], Betw_Cent[2],
                                        benefit, risk, utility]

                    #remove edges between agent and targets
                    G.remove_edge(n, target1)
                    G.remove_edge(n, target2)
                    G.remove_edge(n, target3)


                    #update iteration counter
                    t += 1
                    i += 1

                    #if the new utility score is greater than the current best, make it the best, reset stagnation counter
                    if utility > best:
                        best = utility
                        s = 0
                    else: #if the new utility score is less than the current best, update the stagnation counter
                        s += 1
                        if s == stale[1]: #if stagnation counter is greater than stagnation criteria, stop search
                            stop = True

                            # convert list of lists to a dataframe
                            tmp_data_list = tmp_data_list[0:s]
                            tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)
```

```
                        # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                        #    remove it if you don't care what the 'row number' is in your output
                        tmp_dataframe.index += 1

                        # combine with original dataframe before returning
                        data = pd.concat([data, tmp_dataframe], ignore_index=False)

                        print(s, t, 'too stale')
                        return data

                if t == iterations[1]: #if iteration counter is greater than max iterations, stop search
                    stop = True

                    # convert list of lists to a dataframe
                    tmp_data_list = tmp_data_list[0:t]
                    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                    #    remove it if you don't care what the 'row number' is in your output
                    tmp_dataframe.index += 1

                    # combine with original dataframe before returning
                    data = pd.concat([data, tmp_dataframe], ignore_index=False)

                    print(s, t, 'iteration limit')
                    return data

    print(s, t, 'end of func')
```

```
def screen_by_node_utility(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run):
    '''
    The utility-score screening heuristic.

    Input:
    G: a networkx graph
    Nodes: (list of int): a list of all nodes in the graph
    deg_assort_G (int): degree assortativity of G without agent in network
    stale (list of int): stagnation criteria
    iterations (list of int): the max number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    a: (int) the number of top nodes to keep based of desired measure
    n: (int) number of nodes in the graph
    trial: (int) trial number
    run: (int) run number

    Outputs;
    data: dataframe with utility score information
    total_time: the time required to execute this function (in seconds)
    '''
    data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                   'CC Agent','EC T1', 'BC T1',
                                   'EC T2', 'BC T2',
                                   'EC T3', 'BC T3',
                                   'Benefit', 'Risk', 'Utility'])
    algorithm = "UT"

    start = time.perf_counter()

    #Add agent to network
    G.add_node(n)

    #For each node, calculate the single relationship utility score
    data = one_relationship(G, Nodes, data, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Create a list of the nodes with the top 'a' utility scores
    top_nodes = best_nodes_by_utility(data, a)

    #Perform local search to find the top two-relationship utility scores
    data = get_two_relationship_data(G, data, deg_assort_H, top_nodes, stale, iterations, cost, weights, trial, run, algorithm)

    #Perform local search to find the top three-relationship utility scores
    data = get_three_relationship_data(G, data, deg_assort_H, top_nodes, stale, iterations, cost, weights, trial, run, algorithm)

    #Remove agent from network
    G.remove_node(n)

    end = time.perf_counter()

    total_time = end - start

    return data, total_time
```

Measure Screening Heuristic

```
def best_nodes_by_measure(measure, a):
    '''
    Creates a list of the top 'a' nodes based on a given SNA measure.  'Measure' needs to be the be a dictionary where
    key = node and value = node's measure.

    Input:
    G: a networkx graph
    measure: (dict) result from measure calculation
    a: (int) top number of nodes to keep

    Output:
    best_nodes_measure: a list of the top 'a' nodes ranked by giving measure
    '''
    ##Sorts dictionary by nodes' measure then creates a list of the nodes called initial
    initial = []
    for key, item in sorted(measure.items(), key=lambda x: x[1]):
        initial.append(key)
    initial.reverse()

    ##Creates a list of the top 'a' nodes in initial
    best_nodes_measure = []
    for b in range(0,a):
        best_nodes_measure.append(initial[b])

    return best_nodes_measure;
```

```
def screen_by_measure(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run):
    '''
    The measure screening heuristic.

    Input:
    G: a networkx graph
    Nodes: (list of int): a list of all nodes in the graph
    deg_assort_G (int): degree assortativity of G without agent in network
    stale (list of int): stagnation criteria
    iterations (list of int): the max number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    a: (int) the number of top nodes to keep based of desired measure
    n: (int) the number of nodes in the graph
    trial: (int) trial number
    run: (int) run number

    Outputs;
    data2: dataframe with utility score information
    total_time: the time required to execute this function (in seconds)
    '''
    data2 = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                    'CC Agent','EC T1', 'BC T1',
                                    'EC T2', 'BC T2',
                                    'EC T3', 'BC T3',
                                    'Benefit', 'Risk', 'Utility'])
    algorithm = 'MS'

    start = time.perf_counter()

    #Calculate given measure and then take the top 'a' nodes based on that measure
    Eigen = nx.eigenvector_centrality(G, max_iter = 400)
    best_nodes = best_nodes_by_measure(Eigen, a)

    #Add agent to the network
    G.add_node(n)

    #For each node, calculate the single relationship utility score
    data2 = one_relationship(G, best_nodes, data2, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Perform local search to find the top two-relationship utility scores
    data2 = get_two_relationship_data(G, data2, deg_assort_H, best_nodes, stale, iterations, cost, weights, trial, run, algorithm)

    #Perform local search to find the top three-relationship utility scores
    data2 = get_three_relationship_data(G, data2, deg_assort_H, best_nodes, stale, iterations, cost, weights, trial, run, algorithm)

    #Remove agent from network
    G.remove_node(n)

    end = time.perf_counter()

    total_time = end - start

    return data2, total_time
```

Neighbor-Access Screening Heuristic

```
def neighbors_access_1(G, Nodes, b):
    '''
    Reduces a list one-target scenarios down to those in the top "b" based on the scenario's neighbor-access score.
    The neighbor-access score is the total number of people within two path lengths that can be accessed by the agent
    when the agent develops relationships with the targets.

    Input:
    G: a networkx graph
    Nodes: (list of int) a list of all possible one-target insertion scenarios
    b: (list of int) number of the top scenarios to keep and pass to the heuristic

    Output:
    reduced_singles: a list of tupels
    '''
    neighbor_dict = {}

    for item in Nodes:
        neighbor_dict[item] = len(sorted(G.neighbors(item)))

    new_dict = dict(sorted(neighbor_dict.items(), key=lambda x:x[1], reverse = True))
    ordered_1scenario = list(new_dict.keys())
    reduced_singles = ordered_1scenario[0:b[0]]

    return reduced_singles;
```

```python
def neighbors_access_2(G, pairs, b):
    '''
    Reduces a list two-target scenarios down to those in the top "b" based on the scenario's neighbor-access score.
    The neighbor-access score is the total number of people within two path lengths that can be accessed by the agent
    when the agent develops relationships with the targets.

    Input:
    G: a networkx graph
    pairs: (list of tuples) a list of all possible two-target insertion scenarios
    b: (list of int) number of the top scenarios to keep and pass to the heuristic

    Output:
    reduced_pairs: a list of tuples
    '''
    neighbor_dict = {}

    for item in pairs:
        x = item[0]
        y = item[1]
        neighbors = list(set(sorted(G.neighbors(x))+sorted(G.neighbors(y))))
        if x in neighbors:
            neighbors.remove(x)
        if y in neighbors:
            neighbors.remove(y)
        neighbor_dict[item] = len(neighbors)

    new_dict = dict(sorted(neighbor_dict.items(), key=lambda x:x[1], reverse = True))
    ordered_2scenarios = list(new_dict.keys())
    reduced_pairs = ordered_2scenarios[0:b[1]]
    return reduced_pairs;
```

```python
def neighbors_access_3(G, triplets, b):
    '''
    Reduces a list three-target scenarios down to those in the top "b" based on the scenario's neighbor-access score.
    The neighbor-access score is the total number of people within two path lengths that can be accessed by the agent
    when the agent develops relationships with the targets.

    Input:
    G: a networkx graph
    triplets: (list of tuples) a list of all possible three-target insertion scenarios
    b: (list of int) number of the top scenarios to keep and pass to the heuristic

    Output:
    reduced_triplets: a list of tuples
    '''
    neighbor_dict = {}

    for item in triplets:
        x = item[0]
        y = item[1]
        z = item[2]
        neighbors = list(set(sorted(G.neighbors(x))+sorted(G.neighbors(y))+sorted(G.neighbors(z))))
        if x in neighbors:
            neighbors.remove(x)
        if y in neighbors:
            neighbors.remove(y)
        if z in neighbors:
            neighbors.remove(z)
        neighbor_dict[item] = len(neighbors)

    new_dict = dict(sorted(neighbor_dict.items(), key=lambda x:x[1], reverse = True))
    ordered_3scenarios = list(new_dict.keys())
    reduced_triplets = ordered_3scenarios[0:b[2]]
    return reduced_triplets;
```

```
def na_search_2targets(G, data, deg_assort_H, reduced_pairs, stale_na, iterations_na, cost, weights, trial, run, algorithm):
    '''
    Performs a search on the reduced list of two-target scenarios provided by the neighbor-access screening.
    The search continues until either the desired number of iterations is achieved or a stagnation criteria is met.

    Input:
    G: a networkx graph
    data (dataframe): a dataframe with specific headers
    deg_assort_G (int): degree assortativity of G without agent in network
    reduced_pairs (list of tuples): list of the top b two-target scenarios
    stale_na (list of int): stagnation criteria
    iterations_na (list of int): the max number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    trial: (int) trial number
    run: (int) run number
    algorithm (str): which screening heuristic is being performed

    Outputs;
    data: dataframe with utility score information
    '''
    best = -5 # best utility score set to arbitrarily low number
    t = 0  # iteration counter
    s = 0  # stagnation counter
    i = 0

    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * iterations_na[0]  # empty list; will become list of lists, then finally dataframe

    stop = False
    while stop is False:
        for scenario in reduced_pairs:
            target1 = scenario[0]
            target2 = scenario[1]

            #create edges between the agent and the targets
            G.add_edge(n, target1)
            G.add_edge(n, target2)
            H = ig.Graph.from_networkx(G)

            #calculate the network's degree assortativity with new edges between agent and targets
            deg_assort_w_agent = H.assortativity_degree(directed=False)

            #calculate the betweenness centrality for all nodes
            Betw = H.betweenness(vertices = [target1, target2], directed = False, cutoff = None, weights=None)
            Betw_Cent = []
            for c in range(len(Betw)):
                new = Betw[c]*(2/((n-1)*(n-2)))
                Betw_Cent.append(new)

            #calculate the eigenvector centrality for all nodes
            Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
            #Eig_Cent = normalize(Eig_Cent)

            #calculate the closeness centrality for all nodes
            Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
            #Close_Cent = normalize(Close_Cent)

            #calculate the benefit score using above measures and respective weights out weights list
            benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2])

            #calculate the risk score using above measures and respective weights out weights list
            risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 2*cost

            #calculate the utility score using benefit and risk and respective weights
            utility = weights[4] * benefit - weights[5] * risk

            #write data to dataframe data
            if target1 > target2:
                spot1 = target2
                spot2 = target1
            else:
                spot1 = target1
                spot2 = target2

            tmp_data_list[i] = [trial, run, algorithm, tuple((spot1, spot2, 'NA')), deg_assort_H, deg_assort_w_agent,
                                Close_Cent[0],Eig_Cent[target1], Betw_Cent[0],
                                Eig_Cent[target2], Betw_Cent[1],
                                'NA', 'NA',
                                benefit, risk, utility]

            #remove edges between agent and targets
            G.remove_edge(n, target1)
            G.remove_edge(n, target2)

            #update iteration counter
            t += 1
            i += 1
```

```
            #update iteration counter
            t += 1
            i += 1

             #if the new utility score is greater than the current best, make it the best, reset stagnation counter
            if utility > best:
                best = utility
                s = 0
            else: #if the new utility score is less than the current best, update the stagnation counter
                s += 1
                if s == stale_na[0]: #if stagnation counter is greater than stagnation criteria, stop search
                    stop = True

                    # convert list of lists to a dataframe
                    tmp_data_list = tmp_data_list[0:s]
                    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                    #    remove it if you don't care what the 'row number' is in your output
                    tmp_dataframe.index += 1

                    # combine with original dataframe before returning
                    data = pd.concat([data, tmp_dataframe], ignore_index=False)

                    print(s, t, 'too stale')
                    return data

            if t == iterations_na[0]: #if iteration counter is greater than max iterations, stop search
                stop = True

                # convert list of lists to a dataframe
                tmp_data_list = tmp_data_list[0:t]
                tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                #    remove it if you don't care what the 'row number' is in your output
                tmp_dataframe.index += 1

                # combine with original dataframe before returning
                data = pd.concat([data, tmp_dataframe], ignore_index=False)

                print(s, t, 'iteration limit')
                return data

    print(s, t, 'end of func')
```

```
def na_search_3targets(G, data, deg_assort_H, reduced_triplets, stale_na, iterations_na, cost, weights, trial, run, algorithm):
    '''
    Performs a search on the reduced list of three-target scenarios provided by the neighbor-access screening.
    The search continues until either the desired number of iterations is achieved or a stagnation criteria is met.

    Input:
    G: a networkx graph
    data (dataframe): a dataframe with specific headers
    deg_assort_G (int): degree assortativity of G without agent in network
    reduced_triplets (list of tuples): list of the top b three-target scenarios
    stale_na (list of int): stagnation criteria
    iterations_na (list of int): the max number of interations to be performed
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    trial: (int) trial number
    run: (int) run number
    algorithm (str): which screening heuristic is being performed

    Outputs;
    data: dataframe with utility score information
    '''
    best = -5 # best utility score set to arbitrarily low number
    t = 0  # iteration counter
    s = 0  # stagnation counter
    i = 0

    #initialize temporary data storage to merge with `data` dataframe later; this is done for speed (see optimization notes below)
    tmp_data_list = [None] * iterations_na[1]  # empty list; will become list of lists, then finally dataframe

    stop = False
    while stop is False:
        for scenario in reduced_triplets:
            target1 = scenario[0]
            target2 = scenario[1]
            target3 = scenario[2]

            #add edges between agent and targets
            G.add_edge(n, target1)
            G.add_edge(n, target2)
            G.add_edge(n, target3)
            H = ig.Graph.from_networkx(G)

            #calculate the network's degree assortativity with new edges between agent and targets
            deg_assort_w_agent = H.assortativity_degree(directed=False)
```

```
            #calculate the betweenness centrality for all nodes
            Betw = H.betweenness(vertices = [target1, target2, target3], directed = False, cutoff = None, weights=None)
            Betw_Cent = []
            for c in range(len(Betw)):
                new = Betw[c]*(2/((n-1)*(n-2)))
                Betw_Cent.append(new)

            #calculate the eigenvector centrality for all nodes
            Eig_Cent = H.eigenvector_centrality(directed = False, scale=False, weights=None, return_eigenvalue=False)
            #Eig_Cent = normalize(Eig_Cent)

            #calculate the closeness centrality for all nodes
            Close_Cent = H.closeness(vertices = [n], mode = "all", cutoff = None, weights = None, normalized = True)
            #Close_Cent = normalize(Close_Cent)

            #calculate the benefit score using above measures and respective weights out weights list
            benefit = weights[0] * (Betw_Cent[0] + Betw_Cent[1]+ Betw_Cent[2]) + weights[1] * (Eig_Cent[target1] + Eig_Cent[target2]+ Eig_Cent[target3])

            #calculate the risk score using above measures and respective weights out weights list
            risk = weights[2] * (Close_Cent[0]) + weights[3] * abs(deg_assort_w_agent - deg_assort_H) + 3*cost

            #calculate the utility score using benefit and risk and respective weights
            utility = weights[4] * benefit - weights[5] * risk

            #write data to dataframe data
            tmp_data_list[i] = [trial, run, algorithm, tuple(sorted((target1, target2, target3))), deg_assort_H, deg_assort_w_agent,
                                Close_Cent[0], Eig_Cent[target1], Betw_Cent[0],
                                Eig_Cent[target2], Betw_Cent[1],
                                Eig_Cent[target3], Betw_Cent[2],
                                benefit, risk, utility]

            #remove edges between agent and targets
            G.remove_edge(n, target1)
            G.remove_edge(n, target2)
            G.remove_edge(n, target3)


            #update iteration counter
            t += 1
            i += 1

            #if the new utility score is greater than the current best, make it the best, reset stagnation counter
            if utility > best:
                best = utility
                s = 0
            else: #if the new utility score is less than the current best, update the stagnation counter
                s += 1
                if s == stale_na[1]: #if stagnation counter is greater than stagnation criteria, stop search
                    stop = True

                    # convert list of lists to a dataframe
                    tmp_data_list = tmp_data_list[0:s]
                    tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                    # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                    #    remove it if you don't care what the 'row number' is in your output
                    tmp_dataframe.index += 1

                    # combine with original dataframe before returning
                    data = pd.concat([data, tmp_dataframe], ignore_index=False)

                    print(s, t, 'too stale')
                    return data

            if t == iterations_na[1]: #if iteration counter is greater than max iterations, stop search
                stop = True

                # convert list of lists to a dataframe
                tmp_data_list = tmp_data_list[0:t]
                tmp_dataframe = pd.DataFrame(tmp_data_list, columns=data.columns, index=None)

                # to make index start at 1 instead of 0 (not strictly needed, but makes it match original version)
                #    remove it if you don't care what the 'row number' is in your output
                tmp_dataframe.index += 1

                # combine with original dataframe before returning
                data = pd.concat([data, tmp_dataframe], ignore_index=False)

                print(s, t, 'iteration limit')
                return data

    print(s, t, 'end of func')
```

```
def neighbor_screening(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, b, trial, run, stale_na, iterations_na):
    '''
    Performs the neighbor-access screening heuristic.

    Input:
    G: a networkx graph
    Nodes: (list of int): a list of all nodes in the graph
    all_pairs: (list of tuples): list of all possible two-target scenarios
    all_triplets: (list of tuples): list of all possible three-target scenarios
    deg_assort_G (int): degree assortativity of G without agent in network
    cost: (int) represents the cost to develop a single relationship
    weights: (list of int) list of the six weights used for calculating utility score
    n: (int) number of nodes in the graph
    b: (list of int) the number of one-, two-, and three-target scenarios to keep after screening
    trial: (int) trial number
    run: (int) run number
    stale_na (list of int): stagnation criteria
    iterations_na (list of int): the max number of interations to be performed

    Outputs;
    data3: dataframe with utility score information
    total_time: the time required to execute this function (in seconds)
    '''
    data3 = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                    'CC Agent','EC T1', 'BC T1',
                                    'EC T2', 'BC T2',
                                    'EC T3', 'BC T3',
                                    'Benefit', 'Risk', 'Utility'])
    algorithm = "NA"

    start = time.perf_counter()

    #Perform neighbor-access screening, and keep top percent:
    top_singles = neighbors_access_1(G, Nodes, b)
    top_pairs = neighbors_access_2(G, all_pairs, b)
    top_triplets = neighbors_access_3(G, all_triplets, b)

    #Add agent to the network
    G.add_node(n)

    #For each node, calculate the single relationship utility score
    data3 = one_relationship(G, top_singles, data3, cost, weights, deg_assort_H, trial, run, n, algorithm)

    #Calculate utility score for all top pairs
    data3 = na_search_2targets(G, data3, deg_assort_H, top_pairs, stale_na, iterations_na, cost, weights, trial, run, algorithm)

    #Calculate utility score for all top triplets
    data3 = na_search_3targets(G, data3, deg_assort_H, top_triplets, stale_na, iterations_na, cost, weights, trial, run, algorithm)

    #Remove agent from network
    G.remove_node(n)

    end = time.perf_counter()

    total_time = end - start

    return data3, total_time
```

Functions for Analysis

```
def top_scenarios(data):
    '''
    Creates a list of the top 1, 5, 10, 25, and 50 scenarios based on utility score.

    Input:
    data: dataframe output from one of the screening heuristics or the all-scenarios model

    Output:
    top_ONE: (tuple) the top scoring scenario
    top_FIVE: (list of tuples) the top 5 scoring scenarios
    top_TEN: (list of tuples) the top 10 scoring scenarios
    top_25: (list of tuples) the top 25 scoring scenarios
    top_50: (list of tuples) the top 50 scoring scenarios
    '''
    data_sorted = data.sort_values(by="Utility", ascending = False)
    top_scenarios = data_sorted['Scenario'].to_list()
    tops = []
    for item in top_scenarios:
        tops.append(item)
    top_ONE = tops[0]
    top_FIVE = tops[:5]
    top_TEN = tops[:10]
    top_25 = tops[:25]
    top_50 = tops[:50]

    return top_ONE, top_FIVE, top_TEN, top_25, top_50
```

```python
def one_score(all_ONE, top_ONE):
    '''
    Calculates the accuracy by comparing the top scoring scenario from the all-scenarios model and one of
    the screening heuristics.

    Input:
    all_ONE: (tuple) top scoring scenario from all-scenarios model
    top_ONE: (tuple) top scoring scenario from screening heuristic

    Output: (int) accuracy (0 or 1)
    '''
    if top_ONE == all_ONE:
        score = 1
    else:
        score = 0
    return score
```

```python
def score(all_top, top_num):
    '''
    Calculates the accuracy by comparing a specified number of top scoring scenarios from the all-scenarios model
    and a screening heuristics.

    Input:
    all_top: (list of tuples) specified number of top scoring scenarios from all-scenarios model
    top_num: (list of tuples) specified number of top scoring scenarios from screening heuristic

    Output: (flt) accuracy score
    '''
    correct = 0
    for scenario in range(len(all_top)):
        if top_num[scenario] in all_top:
            correct += 1
    accuracy = correct / len(all_top)
    return accuracy
```

Graph Generation Functions

Erdos Renyi

```python
#G = nx.erdos_renyi_graph(n, p, directed = False)
#H = ig.Graph.from_networkx(G)
```

Relaxed Caveman Graph

l = Number of groups k = number of individuals in group n = l*k p = probability of edge rewire

```python
#G = nx.relaxed_caveman_graph(l, k, p, seed=None)
#H = ig.Graph.from_networkx(G)
```

Barabasi-Albert

m = number of edges to attach from a new node to existing nodes

```python
#G = nx.barabasi_albert_graph(n, m, seed=None, initial_graph=None)
#H = ig.Graph.from_networkx(G)
```

Erdos Renyi Graph

```
master_results = pd.DataFrame(columns = ['Graph', 'Network (n,p,k,m)', 'Trial', 'Run', "All Time",
                                         'NA Top 1', 'NA Top 5', 'NA Top 10', 'NA Top 25', 'NA Top 50', 'NA Time',
                                         'UT Top 1', 'UT Top 5', 'UT Top 10', 'UT Top 25', 'UT Top 50', 'UT Time',
                                         'Ms Top 1', 'Ms Top 5', 'Ms Top 10', 'Ms Top 25', 'Ms Top 50', 'Ms Time',
                                        'Density', 'Diameter', 'Avg Degree', 'Max', 'Min'])


master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                      'CC Agent','EC T1', 'BC T1',
                                      'EC T2', 'BC T2',
                                      'EC T3', 'BC T3',
                                      'Benefit', 'Risk', 'Utility'])

data_filenames = ['ER_trial_1_run_1_data.csv', 'ER_trial_1_run_2_data.csv', 'ER_trial_1_run_3_data.csv',
                  'ER_trial_1_run_4_data.csv', 'ER_trial_1_run_5_data.csv', 'ER_trial_1_run_6_data.csv',
                  'ER_trial_1_run_7_data.csv', 'ER_trial_1_run_8_data.csv', 'ER_trial_1_run_9_data.csv',
                  'ER_trial_1_run_10_data.csv', 'ER_trial_1_run_11_data.csv', 'ER_trial_1_run_12_data.csv',
                  'ER_trial_1_run_13_data.csv', 'ER_trial_1_run_14_data.csv', 'ER_trial_1_run_15_data.csv',
                  'ER_trial_1_run_16_data.csv', 'ER_trial_1_run_17_data.csv', 'ER_trial_1_run_18_data.csv',
                  'ER_trial_1_run_19_data.csv', 'ER_trial_1_run_20_data.csv']
```

```
trial = 1

n = 200
p = .2
k = 'NA'
l = 'NA'

cost = .2
weights = [.5, .5, .5, .5, .5, .5]

a = 20
b = [15, 200, 3920]
stale_na = [10, 10]
iterations_na = [14, 100]
stale = [10, 10] #should be one less than the minimum number of combinations
iterations = [14, 100] #should be one less than the minimum number of combinations

for run in tqdm(range(20)):
    time.sleep(0.5)

    master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                          'CC Agent','EC T1', 'BC T1',
                                          'EC T2', 'BC T2',
                                          'EC T3', 'BC T3',
                                          'Benefit', 'Risk', 'Utility'])

    #Generate Graph
    connected = False
    while connected == False:
        seed = rd.randint(0,1000000)
        G = nx.erdos_renyi_graph(n, p, directed = False, seed=seed)
        connected = nx.is_connected(G)

    H = ig.Graph.from_networkx(G)

    #Create node list and list of all possible two-target and three-target scenarios
    Nodes, all_pairs, all_triplets = node_lists(G)

    #Calculate eigenvector, betweenness, closeness centralities and initial degree assortativity
    deg_assort_H = H.assortativity_degree(directed=False)

    #Calculate utility scores for all possible scenarios
    data4, all_time = all_scenarios(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, trial, run)

    master_data = pd.concat([master_data, data4], ignore_index = True)

    #Determine top x scenarios
    all_ONE, all_FIVE, all_TEN, all_25, all_50 = top_scenarios(data4)

    #Perform Neighbor Screening
    data3, NA_time = neighbor_screening(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, b, trial, run, stale_na, iterations_na)

    master_data = pd.concat([master_data, data3], ignore_index = True)

    #Determine top x scenarios
    NA_ONE, NA_FIVE, NA_TEN, NA_25, NA_50 = top_scenarios(data3)
```

```
    #Determine top x scenarios
    NA_ONE, NA_FIVE, NA_TEN, NA_25, NA_50 = top_scenarios(data3)

    #Calculate accurracy
    NA_top_1 = one_score(all_ONE, NA_ONE)
    NA_top_5 = score(all_FIVE, NA_FIVE)
    NA_top_10 = score(all_TEN, NA_TEN)
    NA_top_25 = score(all_25, NA_25)
    NA_top_50 = score(all_50, NA_50)

    #Perform Utility Screening
    data, UT_time = screen_by_node_utility(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data], ignore_index = True)

    #Determine top x scenarios
    UT_ONE, UT_FIVE, UT_TEN, UT_25, UT_50 = top_scenarios(data)

    #Calculate accurracy
    UT_top_1 = one_score(all_ONE, UT_ONE)
    UT_top_5 = score(all_FIVE, UT_FIVE)
    UT_top_10 = score(all_TEN, UT_TEN)
    UT_top_25 = score(all_25, UT_25)
    UT_top_50 = score(all_50, UT_50)

    #Perform Measure Screening
    data2, MS_time = screen_by_measure(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data2], ignore_index = True)

    #Determine top x scenarios
    MS_ONE, MS_FIVE, MS_TEN, MS_25, MS_50 = top_scenarios(data2)

    #Calculate accurracy
    MS_top_1 = one_score(all_ONE, MS_ONE)
    MS_top_5 = score(all_FIVE, MS_FIVE)
    MS_top_10 = score(all_TEN, MS_TEN)
    MS_top_25 = score(all_25, MS_25)
    MS_top_50 = score(all_50, MS_50)

    #master_data.to_csv('new na test.csv')

    degree = dict(nx.degree(G))

    master_results.loc[len(master_results)+1] = ['ER', (n,p,k,l,a,b,cost,weights,stale,iterations,seed),
                                            trial, run, all_time,
                                            NA_top_1, NA_top_5, NA_top_10, NA_top_25, NA_top_50, NA_time,
                                            UT_top_1, UT_top_5, UT_top_10, UT_top_25, UT_top_50, UT_time,
                                            MS_top_1, MS_top_5, MS_top_10, MS_top_25, MS_top_50, MS_time, nx.density(G),
                                            nx.diameter(G), mean(degree.values()), max(degree.values()),
                                               min(degree.values())]


master_results.to_csv('ER_test_results.csv')
```

Relaxed Caveman Graph

```
master_results = pd.DataFrame(columns = ['Graph', 'Network (n,p,k,m)', 'Trial', 'Run', "All Time",
                                         'NA Top 1', 'NA Top 5', 'NA Top 10', 'NA Top 25', 'NA Top 50', 'NA Time',
                                         'UT Top 1', 'UT Top 5', 'UT Top 10', 'UT Top 25', 'UT Top 50', 'UT Time',
                                         'Ms Top 1', 'Ms Top 5', 'Ms Top 10', 'Ms Top 25', 'Ms Top 50', 'Ms Time',
                                        'Density', 'Diameter', 'Avg Degree', 'Max', 'Min'])


master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                          'CC Agent','EC T1', 'BC T1',
                                          'EC T2', 'BC T2',
                                          'EC T3', 'BC T3',
                                          'Benefit', 'Risk', 'Utility'])

data_filenames = ['CM_trial_1_run_1_data.csv', 'CM_trial_1_run_2_data.csv', 'CM_trial_1_run_3_data.csv',
                    'CM_trial_1_run_4_data.csv', 'CM_trial_1_run_5_data.csv', 'CM_trial_1_run_6_data.csv',
                  'CM_trial_1_run_7_data.csv', 'CM_trial_1_run_8_data.csv', 'CM_trial_1_run_9_data.csv',
                  'CM_trial_1_run_10_data.csv', 'CM_trial_1_run_11_data.csv', 'CM_trial_1_run_12_data.csv',
                  'CM_trial_1_run_13_data.csv', 'CM_trial_1_run_14_data.csv', 'CM_trial_1_run_15_data.csv',
                  'CM_trial_1_run_16_data.csv', 'CM_trial_1_run_17_data.csv', 'CM_trial_1_run_18_data.csv',
                  'CM_trial_1_run_19_data.csv', 'CM_trial_1_run_20_data.csv',]
```

```
trial = 1


n = 200
p = .25 # probability of rewire
k = 10 # number of nodes per group
l = 20 # number of groups

cost = .2
weights = [.5, .5, .5, .5, .5, .5]

a = 20
b = [15, 200, 3920]
stale_na = [50, 50]
iterations_na = [14, 199, 3919]
stale = [50, 50] #should be one less than the minimum number of combinations
iterations = [199, 3919] #should be one less than the minimum number of combinations

for run in tqdm(range(20)):
    time.sleep(0.5)

    master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                          'CC Agent','EC T1', 'BC T1',
                                          'EC T2', 'BC T2',
                                          'EC T3', 'BC T3',
                                          'Benefit', 'Risk', 'Utility'])

    #Generate Graph
    connected = False
    while connected == False:
        seed = rd.randint(0,1000000)
        G = nx.relaxed_caveman_graph(l, k, p, seed=seed)
        G.remove_edges_from(nx.selfloop_edges(G))
        connected = nx.is_connected(G)

    H = ig.Graph.from_networkx(G)

    #Create node list and list of all possible two-target and three-target scenarios
    Nodes, all_pairs, all_triplets = node_lists(G)

    #Calculate eigenvector, betweenness, closeness centralities and initial degree assortativity
    deg_assort_H = H.assortativity_degree(directed=False)

    #Calculate utility scores for all possible scenarios
    data4, all_time = all_scenarios(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, trial, run)

    master_data = pd.concat([master_data, data4], ignore_index = True)

    #Determine top x scenarios
    all_ONE, all_FIVE, all_TEN, all_25, all_50 = top_scenarios(data4)

    #Perform Neighbor Screening
    data3, NA_time = neighbor_screening(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, b, trial, run, stale_na, iterations_na)

    master_data = pd.concat([master_data, data3], ignore_index = True)

    #Determine top x scenarios
    NA_ONE, NA_FIVE, NA_TEN, NA_25, NA_50 = top_scenarios(data3)

    #Calculate accurracy
    NA_top_1 = one_score(all_ONE, NA_ONE)
    NA_top_5 = score(all_FIVE, NA_FIVE)
    NA_top_10 = score(all_TEN, NA_TEN)
    NA_top_25 = score(all_25, NA_25)
    NA_top_50 = score(all_50, NA_50)

    #Perform Utility Screening
    data, UT_time = screen_by_node_utility(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data], ignore_index = True)

    #Determine top x scenarios
    UT_ONE, UT_FIVE, UT_TEN, UT_25, UT_50 = top_scenarios(data)

    #Calculate accurracy
    UT_top_1 = one_score(all_ONE, UT_ONE)
    UT_top_5 = score(all_FIVE, UT_FIVE)
    UT_top_10 = score(all_TEN, UT_TEN)
    UT_top_25 = score(all_25, UT_25)
    UT_top_50 = score(all_50, UT_50)
```

```
    #Perform Measure Screening
    data2, MS_time = screen_by_measure(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data2], ignore_index = True)

    #Determine top x scenarios
    MS_ONE, MS_FIVE, MS_TEN, MS_25, MS_50 = top_scenarios(data2)

    #Calculate accurracy
    MS_top_1 = one_score(all_ONE, MS_ONE)
    MS_top_5 = score(all_FIVE, MS_FIVE)
    MS_top_10 = score(all_TEN, MS_TEN)
    MS_top_25 = score(all_25, MS_25)
    MS_top_50 = score(all_50, MS_50)

    master_data.to_csv(data_filenames[run])

    degree = dict(nx.degree(G))

    master_results.loc[len(master_results)+1] = ['CM', (n,p,k,l,a,b,cost,weights,stale,iterations,seed),
                                            trial, run, all_time,
                                            NA_top_1, NA_top_5, NA_top_10, NA_top_25, NA_top_50, NA_time,
                                            UT_top_1, UT_top_5, UT_top_10, UT_top_25, UT_top_50, UT_time,
                                            MS_top_1, MS_top_5, MS_top_10, MS_top_25, MS_top_50, MS_time, nx.density(G),
                                            nx.diameter(G), mean(degree.values()), max(degree.values()),
                                               min(degree.values())]

master_results.to_csv('CM_trial_1.csv')
```

Barabasi Albert Graph

```
master_results = pd.DataFrame(columns = ['Graph', 'Network (n,p,k,m)', 'Trial', 'Run', "All Time",
                                         'NA Top 1', 'NA Top 5', 'NA Top 10', 'NA Top 25', 'NA Top 50', 'NA Time',
                                         'UT Top 1', 'UT Top 5', 'UT Top 10', 'UT Top 25', 'UT Top 50', 'UT Time',
                                         'Ms Top 1', 'Ms Top 5', 'Ms Top 10', 'Ms Top 25', 'Ms Top 50', 'Ms Time',
                                        'Density', 'Diameter', 'Avg Degree', 'Max', 'Min'])


master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                            'CC Agent','EC T1', 'BC T1',
                                            'EC T2', 'BC T2',
                                            'EC T3', 'BC T3',
                                            'Benefit', 'Risk', 'Utility'])

data_filenames = ['BA_trial_1_run_1_data.csv', 'BA_trial_1_run_2_data.csv', 'BA_trial_1_run_3_data.csv',
                    'BA_trial_1_run_4_data.csv', 'BA_trial_1_run_5_data.csv', 'BA_trial_1_run_6_data.csv',
                  'BA_trial_1_run_7_data.csv', 'BA_trial_1_run_8_data.csv', 'BA_trial_1_run_9_data.csv',
                  'BA_trial_1_run_10_data.csv', 'BA_trial_1_run_11_data.csv', 'BA_trial_1_run_12_data.csv',
                  'BA_trial_1_run_13_data.csv', 'BA_trial_1_run_14_data.csv', 'BA_trial_1_run_15_data.csv',
                  'BA_trial_1_run_16_data.csv', 'BA_trial_1_run_17_data.csv', 'BA_trial_1_run_18_data.csv',
                  'BA_trial_1_run_19_data.csv', 'BA_trial_1_run_20_data.csv']
```

```
trial = 1


n = 50
p = 'NA'
k = 'NA'
m = 3

cost = .2
weights = [.5, .5, .5, .5, .5, .5, .5]

a = 20
b = [15, 200, 3920]
stale_na = [50, 50]
iterations_na = [14, 199, 3919]
stale = [50, 50] #should be one less than the minimum number of combinations
iterations = [199, 3919] #should be one less than the minimum number of combinations

for run in tqdm(range(20)):
    time.sleep(0.5)

    master_data = pd.DataFrame(columns = ['Trial', 'Run', 'Algorithm', 'Scenario', 'DA', 'DA Agent',
                                            'CC Agent','EC T1', 'BC T1',
                                            'EC T2', 'BC T2',
                                            'EC T3', 'BC T3',
                                            'Benefit', 'Risk', 'Utility'])

    #Generate Graph
    connected = False
    while connected == False:
        seed = rd.randint(0,1000000)
        G = nx.barabasi_albert_graph(n, m, seed=seed, initial_graph=None)
        connected = nx.is_connected(G)
```

```
    H = ig.Graph.from_networkx(G)

    #Create node list and list of all possible two-target and three-target scenarios
    Nodes, all_pairs, all_triplets = node_lists(G)

    #Calculate eigenvector, betweenness, closeness centralities and initial degree assortativity
    deg_assort_H = H.assortativity_degree(directed=False)

    #Calculate utility scores for all possible scenarios
    data4, all_time = all_scenarios(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, trial, run)

    master_data = pd.concat([master_data, data4], ignore_index = True)

    #Determine top x scenarios
    all_ONE, all_FIVE, all_TEN, all_25, all_50 = top_scenarios(data4)

    #Perform Neighbor Screening
    data3, NA_time = neighbor_screening(G, Nodes, all_pairs, all_triplets, deg_assort_H, cost, weights, n, b, trial, run, stale_na, iterations_na)

    master_data = pd.concat([master_data, data3], ignore_index = True)

    #Determine top x scenarios
    NA_ONE, NA_FIVE, NA_TEN, NA_25, NA_50 = top_scenarios(data3)

    #Calculate accurracy
    NA_top_1 = one_score(all_ONE, NA_ONE)
    NA_top_5 = score(all_FIVE, NA_FIVE)
    NA_top_10 = score(all_TEN, NA_TEN)
    NA_top_25 = score(all_25, NA_25)
    NA_top_50 = score(all_50, NA_50)

    #Perform Utility Screening
    data, UT_time = screen_by_node_utility(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data], ignore_index = True)

    #Determine top x scenarios
    UT_ONE, UT_FIVE, UT_TEN, UT_25, UT_50 = top_scenarios(data)

    #Calculate accurracy
    UT_top_1 = one_score(all_ONE, UT_ONE)
    UT_top_5 = score(all_FIVE, UT_FIVE)
    UT_top_10 = score(all_TEN, UT_TEN)
    UT_top_25 = score(all_25, UT_25)
    UT_top_50 = score(all_50, UT_50)

    #Perform Measure Screening
    data2, MS_time = screen_by_measure(G, Nodes, deg_assort_H, stale, iterations, cost, weights, a, n, trial, run)

    master_data = pd.concat([master_data, data2], ignore_index = True)

    #Determine top x scenarios
    MS_ONE, MS_FIVE, MS_TEN, MS_25, MS_50 = top_scenarios(data2)

    #Calculate accurracy
    MS_top_1 = one_score(all_ONE, MS_ONE)
    MS_top_5 = score(all_FIVE, MS_FIVE)
    MS_top_10 = score(all_TEN, MS_TEN)
    MS_top_25 = score(all_25, MS_25)
    MS_top_50 = score(all_50, MS_50)

    master_data.to_csv(data_filenames[run])

    degree = dict(nx.degree(G))

    master_results.loc[len(master_results)+1] = ['BA', (n,p,m,a,b,cost,weights,stale,iterations,seed),
                                            trial, run, all_time,
                                            NA_top_1, NA_top_5, NA_top_10, NA_top_25, NA_top_50, NA_time,
                                            UT_top_1, UT_top_5, UT_top_10, UT_top_25, UT_top_50, UT_time,
                                            MS_top_1, MS_top_5, MS_top_10, MS_top_25, MS_top_50, MS_time, nx.density(G),
                                            nx.diameter(G), mean(degree.values()), max(degree.values()),
                                               min(degree.values())]

master_results.to_csv('BA_trial_1.csv')
```

Bibliography

[1] The White House, "Interim National Security Strategic Guidance," Washington, DC, 2021.

[2] The Department of Homland Security, "The DHS Strategic Plan: Fiscal years 2020-2024," Washington, DC, 2020.

[3] P. J. Carrington and J. Scott, "Introduction," in *The SAGE Handbook for Social Network Analysis*, J. Scott and P. J. Carrington, Eds., Los Angeles, SAGE Publications Inc., 2011, pp. 1-8.

[4] M. Tsvetovat and K. Carley, "Knowing the Enemy: A Simulation of Terrorist Organizations and Counter-Terrorism Strategies," in *CASOS Conference*, 2002.

[5] R. C. van der Hulst, "Terrorist Networks: The Threat of Connectivity," in *The SAGE Handbook of Social Network Analysis*, J. Scott and . P. J. Carrington, Eds., Los Angeles, SAGE Publications Inc., 2011, pp. 256-270.

[6] The Mitchell Centre for Social Network Analysis, "Covert Networks," 2021. [Online]. Available: https://www.socialsciences.manchester.ac.uk/mitchell-centre/research/covert-networks/. [Accessed 17 10 2021].

[7] R. C. van der Hulst, "Terrorist Networks: The Threat of Connectivity," in *The SAGE Handbook of Social Network Analysis*, London, SAGE Publications Ltd, 2011, pp. 256-270.

[8] J. Arquilla and D. Ronfeldt, Networks and Netwars: The Future of Terror, Crime and Militancy, Santa Monica, CA: RAND, 2001.

[9] M. Tsvetovat and K. M. Carley, "Structural Knowledge and Success of Anti-Terrorist Activity: The Downside of Structural Equivalence," *Journal of Social Structure*, vol. 6, no. 2, 2005.

[10] K. M. Carley, "Estimating vulnerabilities in large covert networks," Carnegie-Mellon University, Pittsburg, PA, 2004.

[11] Joint Chiefs of Staff, "Joint Publication 3-25: Countering Threat Networks," 21 December 2016. [Online]. Available: https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp3_25.pdf.

[12] P. J. Carrington and J. Scott, "Introduction," in *The SAGE Handbook for Social Network Analysis*, London, SAGE Publications Ltd, 2011, pp. 1-8.

[13] S. P. Borgatti, M. G. Everett and J. C. Johnson, Analyzing Social Networks, 2nd ed., London: SAGE Publications Ltd, 2018.

[14] M. O. Jackson, Social and Economic Networks, Princeton, NJ: Princeton University Press, 2008.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature,* vol. 393, no. 6684, pp. 440-442, 1998.

[16] M. E. Newman and M. Girvan, "Finding and evaluating community struture in networks," *Physical Review,* vol. 69, no. 2, p. 026113, 2004.

[17] M. E. J. Newman, "Modularity and community struture in networks," *Proceedings National Academy of Sciences of the United States of America,* vol. 103, no. 23, pp. 8577-8582, 2006.

[18] D. Zinoviev, Complex Network Analysis, Raleigh, NC: The Pragmatic Bookshelf, 2018.

[19] M. E. J. Newman, "Mixing patterns in networks," *Physical Review E,* vol. 67, no. 2, p. 026126, 2003.

[20] L. C. Freeman, "Centrality in Social Networks: Conceptual Clarification," *Social Networks,* vol. 1, no. 3, pp. 215-239, 1978/1979.

[21] S. P. Borgatti, "Centrality and Network Flow," *Social Networks,* vol. 27, no. 1, pp. 55-71, 2005.

[22] K. Stephenson and M. Zelen, "Rethinking Centrality: Methods and Examples," *Social Networks,* vol. 11, no. 1, pp. 1-37, March 1989.

[23] P. Bonacich, "Power and Centrality: A Family of Measures," *American Journal of Sociology,* vol. 92, no. 5, pp. 1170-1182, March 1987.

[24] L. Katz, "A New Status Index Derived from Sociometric Analysis," *Psychometrika,* vol. 18, no. 1, pp. 39-43, 1953.

[25] J. M. Fletcher and T. Wennekers, "From Structure to Activity: Using Centrality Measures to Predict Neuronal Activity," *International Journal of Neural Systems,* vol. 28, no. 2, p. 1750013, 2017.

[26] X. Qin, R. D. Duval, K. Christensen, E. Fuller, A. Spahiu, Q. Wu, Y. Wu, W. Tang and C. Zhang, "Terrorist Networks, Network Energy and Node Removal: A New

Measure of Centrality Based on Laplacian Energy," *Social Networking,* vol. 2, pp. 19-31, 2013.

[27] J. D. Guzman, R. F. Deckro, M. J. Robbins, J. F. Morris and N. A. Ballester, "An Analytical Comparison of Social Network Measures," *IEEE Transactions on Computational Social Systems,* vol. 1, no. 1, pp. 35-45, 2014.

[28] S. P. Borgatti, K. M. Carley and D. Krackhardt, "On the robustness of centrality measures under conditions of imperfect data.," *Social Networks,* vol. 28, pp. 124-136, 2006.

[29] A.-L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science,* vol. 286, no. 5439, pp. 509-512, 1999.

[30] P. Erdos and A. Renyi, "On Random Graphs," *Publicationes Mathematicae,* vol. 6, pp. 290-297, 1959.

[31] D. J. Watts, "Networks, Dynamics, and the Small-World Phenomenon," *American Journal of Sociology,* vol. 105, no. 2, pp. 493-527, 1999.

[32] S. Fortunato, "Community Detection in Graphs," *Physics Reports,* vol. 486, no. 3-5, pp. 75-174, 2010.

[33] T. Diviak, "Key aspects of covert networks data collection: Problems, challenges, and opportunities," *Social Networks,* 2019.

[34] V. Asal and R. K. Rethemeyer, "Researching terrorist networks," *Journal of Security Education,* vol. 1, no. 4, pp. 65-74, 2006.

[35] K. M. Carley, J.-S. Lee and D. Krackhardt, "Destabiliing Networks," *Connections,* vol. 24, no. 3, pp. 79-92, 2002.

[36] K. M. Carley, M. Dombroski, M. Tsvetovat, J. Reminga and N. Kamneva, "Destabilizing Dynamic Covert Networks," in *Proceedings of the 8th International Command and Control Research and Technology Symposium,* National Defense War College, Washington, DC, 2003.

[37] J. L. Geffre, R. F. Deckro and S. A. Knighton, "Determining Critical Members of Layered Operational Terrorist Networks," *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology,* vol. 6, no. 2, pp. 97-109, 2009.

[38] C. A. J. Johnstone, "A Risk Based Approach to Node Insertion within Socail Networks," 2015. [Online]. Available: https://scholar.afit.edu/etd/118.

[39] T. Kean and L. Hamilton, "The 9/11 Commission Report: Final Report of the National Commission on Terrorist Attacks Upon the United States," Government Printing Office, Washington, 2004.

[40] A. A. Hagberg, D. A. Schult and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, 2008.

[41] J. Banks, J. S. Carson II, B. L. Nelson and D. M. Nicol, Discrete-Event System Simulation, 5th ed., Upper Saddle River, NJ: Prentice Hall, 2010.

[42] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry,* vol. 40, no. 1, pp. 35-41, March 1977.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
24-03-2022	Master's Thesis	September 2020 - March 2022

4. TITLE AND SUBTITLE
Screening Heuristics for the Evaluation of Covert Network Node Insertion Scenarios

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Pekarek, Andrew E, MAJ

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENS-MS-22-M-161

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A. Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
The majority of research on covert networks uses social network analysis (SNA) to determine critical members of the network to either kill or capture for the purpose of network destabilization. This thesis takes the opposite approach and evaluates potential scenarios for inserting an agent into a covert network for information gathering purposes or future disruption operations. Due to the substantial number of potential insertion scenarios in a large network, this research proposes three screening heuristics that leverage SNA measures to reduce the solution space before applying a simple search heuristic.

15. SUBJECT TERMS
covert networks, screening heuristics, social network analysis, terrorist networks, node insertion

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Michael Garee, PhD, AFIT/ENS
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	U	124	(937) 255-3636 x4510