2009

Greedy methods for approximate graph matching with applications for social network analysis

Partha Basuchowdhuri
Louisiana State University and Agricultural and Mechanical College

GREEDY METHODS FOR APPROXIMATE GRAPH MATCHING WITH APPLICATIONS FOR SOCIAL NETWORK ANALYSIS

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science

in
The Department of Computer Science

by
Partha Basuchowdhuri
B.E., Bengal Engineering College, 2003
M.S.E.E., Louisiana State University, 2008
May 2009

Acknowledgements

I am very grateful to those people who have made this thesis possible and to those who have made my experience in the graduate school one that I will always remember fondly.

I want to deeply thank my primary advisor Dr. Jianhua Chen for her patient guidance throughout the period of my Master's study. It was her Data mining course that inspired me to work on this topic. Without her patience, encouragement, expertise, this work would not have been possible. She challenged me to do my best in learning, thinking, writing, research and personal growth and her support provided me with the motivation and determination to complete this degree.

I would also like to specially thank Dr. Peter P. Chen for providing me with financial support during this work and allowing me the privilege to be a part of his team. My deepest thanks go to my advisor and mentor Dr. Sukhamay Kundu, who has always been an inspiration to me. I would also like to thank Jian Xu, Dr. Bijaya Karki and Dipesh Bhattarai for their professional help.

I am also grateful to my loving wife Sreemoyee, whose unconditional care and support during this period has made this thesis possible. I am also thankful to my parents, my brother and all other family members for their encouragement. I would also like to thank all the people from Baton Rouge, who have helped me, cared for me and encouraged me during this period.

I would like to dedicate this work to two of my late relatives, late Hashirani De and late Bidhurani Bishnu, whom I miss dearly.

Table of Contents

List of Tables

List of Figures

Abstract

In this thesis, we study greedy algorithms for approximate sub-graph matching with attributed graphs. Such algorithms find one or multiple copies of a sub-graph pattern from a bigger data graph through approximate matching. One intended application of sub-graph matching method is in Social Network Analysis for detecting potential terrorist groups from known terrorist activity patterns. We propose a new method for approximate sub-graph matching which utilizes degree information to reduce the search space within the incremental greedy search framework. In addition, we have introduced the notion of a "seed" in incremental greedy method that aims to find a good initial partial match. Simulated data based on terrorist profiles database is used in our experiments that compare the computational efficiency and matching accuracy of various methods. The experiment results suggest that with increasing size of the data graph, the efficiency advantage of degree-based method becomes more significant, while degree-based method remains as accurate as incremental greedy. Using a "seed" significantly improves matching accuracy (at the cost of decreased efficiency) when the attribute values in the graphs are deceptively noisy. We have also investigated a method that allows to expand a matched sub-graph from the data graph to include those nodes strongly connected to the current match.

1 Introduction

In this thesis, we will be presenting the theories to track down an individual terrorist suspect or a group of potential terrorists, with help of graph mining techniques. We will also be presenting the experimental results by applying these techniques on synthetic and real data and also compare them with the results of related previous works. Through this work we want to improve on the manual process of the terrorist network analysis and help those who need to analyze a huge amount of similar data in an automated manner with little guidance and also within a short period of time.

In a social structure of a geographical domain, terrorists can exist without making their intentions public, just like other non-terrorist individuals. So our goal is to find the individual terrorists or a group of terrorist suspects from a geographical domain, for which we have the knowledge of the certain characteristics of every individual and some features of their communications or interactions with others. Selection of these features can be key to the mining process and are selected by experts, who may or may not be computer scientists.

For a larger database it is easier to use structures like graph or lattice rather than trees. So in this case we have represented the data available in terms of nodes i.e, the individuals present in the geographic domain and the edges i.e, the relationships or transactions between the individuals. Ideally it should look like a social network graph which has high number of nodes and may or may not be fully connected. We will mention this graph as the data graph in this thesis, throughout.

We have developed our theories with practical scenarios in mind. We have realized three potential scenarios and have tried to find solutions for them. It is essentially understandable that trying to find matches for terrorists cannot be done by exact matching and hence we have used weighted inexact matching to detect individuals or a pattern.

The three practical scenarios based on the extent of apriori knowledge can be as follows:

There can be full knowledge of a terrorist network, which has taken refuge in a certain geographic domain. This means that we already have a complete pattern graph for which we have to find a match.

There can be partial knowledge of a terrorist network, which has taken refuge in a certain geographic domain. This means we have partial information about a pattern graph. In this case certain nodes and edges could only be matched. It is necessary to understand that the individuals or nodes with apriori knowledge should have complete feature information and should not be partial. Other techniques are applied to find out the other members of the network, which can not be found by inexact matching.

There can be absolutely no knowledge about the terrorist individuals. Only some generalized information may be known. This can be considered as a special variation of the first case, where some forced apriori knowledge may be created based on the feature selections of the nodes by the experts. As this is a special case of the first scenario, we focus mainly on the first two scenarios.

The main way to identify the terrorists is their communications, interactions or transactions. In other words, the edge information can be key to a successful match. We extend the edge information in terms of degree information to identify a potential suspect. Degree information and other key definitions mentioned in Section 2.1, have been central to our idea of detecting terrorist networks and its key players.

1.1 Overview of Social Network Analysis

In the following sections the purpose of Social Network Analysis and its use among the researchers will be discussed.

1.1.1 Purpose of Social Network Analysis

Social Network Analysis makes way for a much needed detailed study of organized crimes such as terrorism, drug-trafficking and other gang related crimes, corporate level fraud and money laundering. Other than terrorist network, it is known that criminals make their group to perform one form of illegal operation and such groups remain in constant communications with other related groups. Although terrorist networks are thought to be primary purpose of Social Network Analysis, similar criminal networks like drug network gangs or the “Mafia” gangs can also be traced. Even SNA can be used to find patterns from some social network group to trace people with similar intentions or similar goals.

While investigating a social or a criminal network, the goals of the investigators are to find answers for the following questions [1] [2]:

- What are the sub-groups in the network and how many of them are present?

This gives an idea of how many sub-groups are present in the network and if they have similar goals.

- How the interaction between the groups is made?

This mainly says about the patterns of the interactions between the subgroups, which may also be used later as apriori knowledge for cases where no real priori information is available.

- What are the roles of the members of a network? How would a member's removal affect the network?

This leads to finding out and studying the roles of individual members from their relationships and interactions, which is later needed to disrupt the functionality of the network most effectively.

- What is the overall structure of the network?

This will show an overall picture of all the sub-group, their functionality, interactions between themselves and the overall functionality and organization of the network.

- How does the information flow in the network?

Knowledge of these organizational features will help exposing the bottlenecks of a social network and will lead to important advancements in criminal investigation.

1.1.2 Literature Review of Social Network Analysis

Intelligence bureaus have been using social network analyzing tools to get a more automated, faster approach to track criminals [3] [4]. Saddam Hussein's capture was facilitated by Social Network Analysis tools. First the military officials created a network containing Saddam Hussein's family members and close relatives. Then they tracked down close tribal or relative friends to eventually find Saddam Hussein [5]. Social Net work

analysis originated from social science research as a set of analytical tools that could map the members and their relationships and extract means of collaborations between important individuals or groups. The techniques were originally designed to recognize social patterns in a social networks [6] and are pertinent for analyzing criminal or terrorist networks [1] [2] [7]. In a few literatures some centrality measure like degree, closeness, betweenness and eigen vectors have between defined so that it could identify the importance of a member [1] and can also identify the leaders, gatekeepers and outliers in a network [8]. The centrality factors are mainly derived from a literature by Freeman [9]. Degree information has very important in analyzing social network. Such an example is finding the central individuals in a price-fixing conspiracy networks in the electrical equipment industry [10].

1.1.3 Existing Network Analysis Tools

Criminal network analysis tools are categorized into three generations [11]:

First Generation: Manual Approach

Anacpapa chart of Harper and Harris [12] is considered one of them. This process involved construction of an association matrix and its link chart and the criminal was placed at the centre of the link chart. Then the structure of the graphical portrayal was studied to find patterns of interest. Krebs [3] manually constructed a terrorist network comprising the 19 hijackers involved in the 9/11 attack. He examined publicly released data from the major newspapers to gather information about relationships of those involved with others and among themselves. He manually constructed an association matrix to integrate these relations and drew a terrorist network showing all possible patterns of interactions between the terrorists based on the matrix.

Manual approach is time consuming and becomes ineffective when the number of nodes in the networks is even in the order of 100s.

Second generation: Graphics-Based Approach

Examples of second generation social network tools are Analyst’s Notebook, Netmap and Watson. These can produce an automatic graphical representation of the networks.

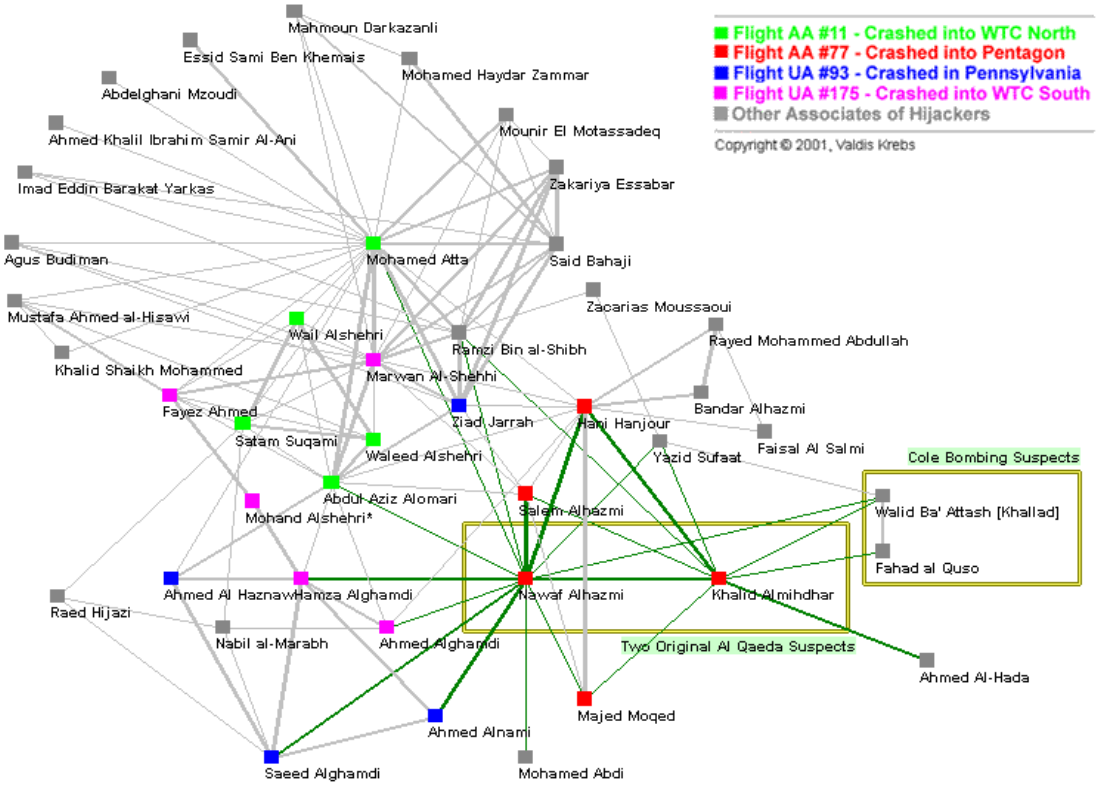

Figure 1. The Terrorist Network surrounding 19 hijackers on Sep 11, 2001 (Source: http://www.orgnet.com)

Analyst's Notebook has been widely employed by law enforcement agencies in USA and Netherlands. It depends on a human analyst to detect the relationships from a link chart, which can be automatically generated from stored relational data. Different entities in a network can be differentiated by different icons and the user could drag the icons to rearrange the network for ease of visualization.

Netmap lays out all the entities on the perimeter of a circle and connects them with straight lines to represent links between them. This software has been adopted in the FinCEN system of the U.S. Treasury Department to analyze financial transaction data to discover money laundering incidents.

Although visualization is an added factor in second-generation social analysis tools the responsibility of analysis is still on the human analysts as these tools offer little structural capabilities.

Third generation: Structural Analysis Approach

Presently although there have been advancements in terms of approaches and ongoing research, still no SNA tool is fit to be categorized as a third generation tool.

Social network analysis is used in order to discover relations and interaction and find patterns among the social entities in studies related to sociology [13]. Other than criminal or terrorist networks Social network analysis tool has also been employed for organizational behavior, inter-organization relations, citation patterns, computer mediated communication and many others [14]. Studies involving criminal network, evidence mapping in fraud and conspiracy cases are more recent.

1.2 Applying Sub-Graph Mining to Social Network Analysis

Introduction of Sub-Graph mining for Social network analysis, although introduced very recently, has been very popular among the researchers.

1.2.1 Why Sub-Graph Mining?

Although the formal definition of sub-graph is given in chapter 2, it could be said that a graph (S) is a called a sub-graph of a graph (G) when S consists of only vertices and edges those are part of G. G can also be referred to as super graph of S. In other terms, S is a part of the super graph G.

In a social network, all of its nodes could represent individuals, who are part of that network. But a criminal group may only be a part of that social network, where some individuals residing within that network may be performing acts of criminal motive with direct or indirect help of other individuals they are related to within that network. Hence by definition of sub-graph, the criminal network would be a sub-graph, consisting of the individuals performing crimes as nodes and its super-graph would be a graph consisting of all individuals of the social network as nodes. So, if we have total knowledge of the social network in terms of the features of individuals and features of relationship between two individuals, based on the important attributes that could distinguish the criminals and non-criminals, with a prior knowledge of certain criminal individuals and their relationships we could try to find a match of that criminal group inside the social network and successfully find them.

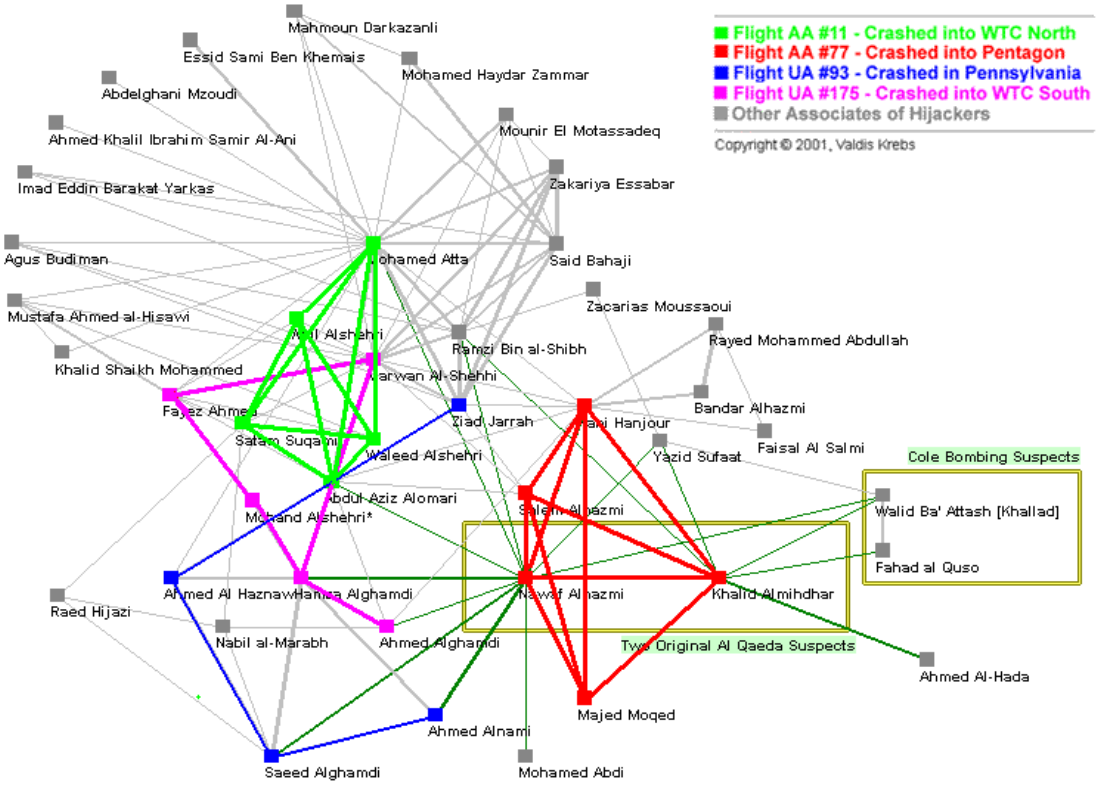

Figure 2. Sub-graphs of 9/11 hijacker network based on the flights hijacked

In Figure 2, using Figure 1 as the source, connected sub-graphs with vertices and edges of same colors have been drawn. These connected sub-graphs depict group of hijackers which hijacked the flights. The groups of hijackers have been represented by different colors based on the flight they took control of. This is a simple basis of representing sub-graphs. Sub-graphs can be much more complex based on number of overlapping vertices and edges. For example, the node labeled “Mohamed Atta” has relationships with nodes of all the colored sub-graphs and in a different representation it could be part of an extended version of all of these sub-graphs.

1.2.2 Literature review: Using Sub-Graph Mining Techniques for SNA

There have been many instances where graph mining or generally speaking, data mining has been used to analyze a social network. It has been used not only for analyzing criminal networks, but also customer networks for companies, IMDB networks, DBLP co-author

networks, etc. have been subject to testing mining techniques in purpose of social network analysis.

Application of data mining to viral marketing was proposed in [15] viewing customers as nodes of a social network. Their influence on each other was represented by a Markov Random model. Based on important features of a social network data available in CVS repositories have been analyzed [16]. Probabilistic data mining models like probabilistic relational models (PRM) and relational probability trees (RPT) have been used to analyze data from IMDB [17]. In [18] mining is done on network graphs in order to find N-cliques with “friend of a friend” relaxation instead of strict direct links and used a popular graph isomorphism technique to find frequent sub-graphs. Multiple relation based community mining has been mentioned in [19], where they defined how an attribute could gain or lose importance based on acquired knowledge.

2 Theoretical Background and Preliminaries

In this section the theories needed to build up this thesis will be discussed.

2.1 Important Definitions

The basics of graph, graph-mining and social network analysis terms are discussed in the following part in terms of definitions.

2.1.1 Definitions Related to Graphs and Graph Mining

A **graph** *G* is an ordered pair *G:=(V,E)* that maintains following conditions,

- *V* is a set, whose elements are called **vertices** or nodes.
- *E* is a set of unordered pairs of vertices, called **edges**.

The order of a graph is denoted by $|V|$ and size by $|E|$.

In graph theory, **degree** D_i of a vertex i can be defined as the number of edges incident on that vertex and can be represented as,

$$D_i = \sum_{j=1}^{V} n_{ij} \begin{cases} n_{ij} = 0, \ when\ ij\ \ doesnot\ \ exist \\ n_{ij} = 1, \ when\ ij\ \ \ exists \\ n_{ii} = 0 \end{cases},$$

where, *ij* is the edge between node *i* and node *j* and *V* is the number of vertices present in the graph.

In a special case, when all ij exist and all vertices have $D_i = V - 1$, the graph is called a **complete graph or fully connected graph**. In fully connected graphs, all the vertices are connected to every other vertex in the graph.

Say a graph *G* consists of *n* nodes. Then,

$V = \{v_1, \ v_2, \ ...., \ v_n\}$ and *E* is a set of pairs of the form $(v_i, \ v_j)$, which denotes edges between vertex *i* and vertex *j*. If $(v_i, \ v_j)$ is considered to be the same as $(v_j, \ v_i)$, then the pairs of vertices are unordered pairs and the graph G is called an **undirected graph**.

If (v_i, v_j) is considered to be the different from (v_j, v_i), depending on the direction of the edge, then the pairs of vertices are called ordered pairs and the graph G is called a **directed graph**. In our notation used in this thesis $(v_i, v_j) \Leftrightarrow (i, j) \Leftrightarrow ij$, i.e., we will often refer the edge between v_i and v_j as *ij*. When edges are undirected, they simply indicate the existence of a relation between two vertices. On the other hand, directed edges are used when relations between vertices are considered in a not symmetric way.

Graphs can be of two types based on the number of relations characterized by an edge between two vertices. Firstly, an edge could just bear status of one relationship, i.e, where or not that relationship is present between those two vertices. This kind of relationship is called simple relationship and the graphs having only simple relationships are called a **simplex graph**. Secondly, an edge could bear a label that has a vector of multiple features that characterize the relationship between those two vertices. These relations are called multiplex relations and the graphs with only these kinds of relationships are called **multiplex graph**s. In section 2.3, it can be clearly seen that the database we are dealing with can be represented as a multiplex graph.

A **subgraph** (G_S) of graph *G* is a graph whose sets of vertices (V') and edges (E') are subsets of the set of vertices and edges of *G*. It can be written as $G_S := (V', E')$, where $V' \subseteq V$, and $E' \subseteq E$. *G* is called a **super-graph** of G_S. A graph *G* contains another graph G_S if some subgraph of *G* is G_S or is isomorphic to G_S. In complexity theory, **subgraph isomorphism** is a decision problem that is known to be NP-complete.

A **clique** in a graph is a set of pairwise adjacent vertices, or an induced subgraph which is a fully connected graph. The **clique problem** is the problem of determining whether a graph contains a clique of at least a given size *k*. Once we have located *k* or more vertices which form a clique, it's trivial to verify that they do, which is why the clique problem is in NP hard. The corresponding problem of finding the largest clique in a graph is called the **maximum clique problem**.

Graph vertices and edges sometime contain information. When this information is a simple label (i.e. a name or number) the graph is called a **labeled graph**. In other more complex representation, vertices and edges may contain some more information like vertex and edge attributes. In this case the graph is called an **attributed graph**. This concept is further elaborated by distinguishing between vertex-attributed (or weighted graphs) and edge-attributed graphs.

2.1.2 Definitions Related to Social Network Analysis

Analysis of a social network mainly focuses on relationships and interactions between members of the network. The labels depicting information about their relationship are used to identify central members or partition a network into several smaller subgroups based on their functionality.

The term **geodesic** is often used in social network analysis studies to denote the shortest length between any two nodes in the network.

Several centrality measures [9] can be used to initially detect the members and then identify their roles within a network.

Degree centrality measures how active a particular node is, in terms of its relationships with other nodes. It can be defined as the number to edges connecting to a node *x*, and can be described as

$$C_D(x) = \sum_{i=1}^{n} a(x,i),$$

where *n* is the total number of nodes in the network and $a(x,i)$ is a binary variable indicating whether or not, an edge is present between nodes *x* and *i*. A network member with high degree can be identified as a leader or a "hub" in the network.

Betweenness centrality measures the extent to which a particular node lies between other nodes in the network. The betweenness of a node x can be defined as the number of geodesics passing through it and can be described as

$$C_B(x) = \sum_{x}^{n} \sum_{j}^{n} g_{ij}(x),$$

where $g_{ij}(x)$ indicates whether the shortest path between two random nodes of that network i and j passes through the node x. A member with high betweenness may act as a gatekeeper or "broker" in a network for communications or flow of resources (part of weapon, chemicals, maps etc.).

Closeness centrality is the sum of the length of geodesics between a particular node x and all the other nodes in the network. It actually measures how far away one node is from other nodes and is also sometimes referred to as **farness centrality**. It can be described as:

$$C_C(x) = \sum_{i=1}^{n} l(x,i),$$

where $l(x,i)$ denotes the length of the shortest path between nodes *x* and *i*.

Sometimes, in some of the literatures, to preserve the true essence of closeness, it is depicted by $C_C(x) = \frac{1}{\sum_{i=1}^{n} l(x,i)}$, and described as inverse of closeness. But to keep thing simple, in this thesis closeness and farness will be treated as same term and will be depicted by the previous equation.

From these basic centralities, new centralities have been derived and are being mentioned in literatures [20]. Complex networks can be described by their **network efficiency centrality** $C_{eff}(G)$, which is a measure to quantify how efficiently the nodes of a network exchange information. It is important to notice that efficiency of a network is not described in terms of a single node. It can be described as

$$C_{eff}(G) = \frac{1}{n(n-1)} \sum_{i \neq j \in G} \frac{1}{l(i,j)}$$

Position role centrality can be defined as the difference of the efficiency of the undisturbed network and the efficiency of the network if that node is removed. This centrality can describe how significant a node is in terms of network efficiency and hence realize their roles. It can be described as

$$C_{PR}(i) = C_{eff}(G) - C_{eff}(G - V_i),$$

where $C_{eff}(G - V_i)$ is the efficiency of the network with the node *i* removed from the graph *G*. Say, if *i* is more significant role (i.e., more connected) than *j*, $C_{eff}(G - V_i)$ value with be lesser than $C_{eff}(G - V_j)$. Hence, $C_{PR}(i)$ will be greater than $C_{PR}(j)$.

Dependence centrality represents how much one node is dependent on other nodes in a network. Under a circumstance where a node *u* wants to communicate with *w* and do it via *v*, we could calculate dependence factor of *u* on *v* ($\zeta_{(u.v)}(w)$). It can be described as

$$\zeta_{(u.v)}(w) = \frac{occurence(u,v)}{l(u,v) \times path(u,w)},$$

where, $occurence(u,v)$ = the number of times (shortest paths) the node *u* uses node *v* for communicating with each other,

$path(u,w)$ = the number of shortest paths between nodes *u* and *w*, and

$l(u,v)$ = the geodesic distance between nodes *u* and *w*.

Dependence centrality of a node can be represented as the degree to which a node *u* must depend upon another node *v*, to relay its message along the geodesics to and from all other reachable nodes in the network. For a network with *n* nodes the dependence centrality of *u* and *v* can be found by the following equation:

$$C_{dep(u,v)} = 1 + \sum_{w=1}^{n} \zeta_{(u,v)}(w), \quad u \neq v \neq w$$

These definitions, which identify the nodes of a graph in terms of social network analysis, help us to recognize the importance and the role of a node in a network.

2.2 Exact and Inexact Graph Matching

In pattern recognition problems, where it is required to find a match of a smaller pattern graph (G_P) inside a bigger data graph (G_D), the procedure involves comparing them to check whether there exists a copy of the pattern graph within the data graph.

The graph matching problem can be stated as follows: Given two graphs $G_P = (V_P, E_P)$ and $G_D = (V_D, E_D)$, with $|V_P| = |V_D|$, the problem is to find a one-to-one mapping $f : V_P \rightarrow V_D$ such that $(v_i,\ v_j) \in E_P$ iff $(f(v_i),\ f(v_j)) \in E_D$. When such a mapping *f* exists, this is called an isomorphism, and G_P is said to be isomorphic to G_D. This type of problems is said to be exact graph matching.

The term inexact applied to some graph matching problems means that it is not possible to find an isomorphism between the two graphs to be matched. In a criminal network a behavioral change or change in affiliations could lead to slightly different features than recorded in the pattern graph, i.e., the prior knowledge. Also there are possibilities where data is incomplete because of practical data acquisition difficulties and hence an exact matching is not possible. In these cases no isomorphism can be expected between both graphs, and the graph matching problem does not consist in searching for the exact feature matching of vertices or edges of a pattern graph with vertices or edges of the data graph, instead finding the best possible matching between them will lead to a better solution. These circumstances give rise to a class of problems known as inexact graph matching. In this problem, the matching aims at finding a non-bijective correspondence between a data graph and a model graph. In the following we will assume $|V_P| < |V_D|$.

The best correspondence of a graph matching problem is defined as the optimum of some objective function which measures the similarity between matched vertices and edges. In an inexact graph matching problem we have $|V_P| < |V_D|$, the goal is to find a mapping $f' : V_P \to V_D$ such that $(v_i,\ v_j) \in E_P$ iff $(f'(v_i),\ f'(v_j)) \in E_D$. This corresponds to the search for a small graph within a big one. An important sub-type of these problems are sub-graph matching problems, in which we have two graphs $G = (V, E)$ and $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, and in this case the aim if to find a mapping $f' : V' \to V$ such that $(v_i,\ v_j) \in E'$ iff $(f'(v_i),\ f'(v_j)) \in E$. When such a mapping exists, it is called a subgraph matching or subgraph isomorphism.

In inexact or approximate graph matching, where we have $|V_P| \leq |V_D|$, the complexity is proved in [22] to be NP-complete. Similarly, the complexity of the inexact sub-graph problem is equivalent in complexity to the largest common sub-graph problem, which is known to be also NP-complete.

2.3 Attributed Graph Representation of the Data

In our pattern graph and data graph the vertices and edges are represented by a vector of features that represents characteristics of that vertex or that edge. The label or the feature vector can be represented as $\{x_1,\ x_2,\ ....,\ x_n\}$ where there are *n* attributes. Initially we ran our experiments on a smaller randomly generated synthetic dataset where value of x_i was binary. But in the pre-processed terrorist database x_i has been represented by decimal numbers.

Each vertex has the same number of attributes or features for the feature vector. The sets of vertex feature vectors can be represented by $F_V = \{x_{V_1},\ x_{V_2},\ ....,\ x_{V_p}\}$ and similarly edge feature vectors can be represented by $F_E = \{x_{E_1},\ x_{E_2},\ ....,\ x_{E_q}\}$ where *p* may not be equal to *q*.

Although the graph may not consist of edges between some of it vertices, the graph is considered as a complete graph and the feature vectors of the edges, which do not exist, can be represented by an all zero string of length *q*.

2.4 The Problem Statement

The central goal of this project has been to detect inexact matches for some known patterns within a network. The other goals are to detect roles of individuals in the network using the definitions of SNA. Also when no or partial prior knowledge is available about the patterns we could aim to extract useful information to find patterns based on some pre-determined criteria.

PRIMARY GOAL

In an ideal situation we will have information about criminal groups which is a part of a social network. The criminal groups will be represented in form of attributed graphs. Our primary goal will be to find matches of those pattern graphs from the graph that represents the social network. Since the values of the attributes that identify a criminal may change slightly, we use inexact match for the process.

The attributed graph can be represented as $G = (V,\ A_V,\ E)$ where *V* is the number of the vertices, A_V is the attribute vector for the vertex *V*. Say, *V* has *n* number of vertices and can be represented as $1 \times n$ matrix. If A_V consists of *p* number of features in its feature vector, it can also be represented by an adjacency matrix with dimension $n \times p$. *E* being the edges or the relationships between the corresponding edges, it can be represented by a $n \times n \times q$ matrix where *q* is the number of features representing an edge.

Given a pattern graph $G_P = (V_P, A_{V_P}, E_P)$ and a data graph $G_D = (V_D, A_{V_D}, E_D)$, where numbers of nodes in the data graph are much larger than that of in the pattern graph, we try to find a subgraph of G_D referred to as $G_{D,S}$, where $G_P \equiv G_{D,S}$. Our goal is to find a mapping where the mapping function is $f : V_P \rightarrow V_D$, such that for every $(v_i,\ v_j)$, there is a distance function for vertices (V_{dis}) and edges (E_{dis})which can be denoted as

$$E_{dis} \Rightarrow \sum_{(v_i, v_j)\ in\ (V_P \times V_P)} dis[E_P(v_i, v_j),\ E_D(f(v_i), f(v_j))]$$

and $V_{dis} \Rightarrow \sum_{i=1}^{n} dis[A_{V_P}(v_i),\ A_{V_D}(f(v_i))]$, where there are *n* number of vertices in the pattern graph.

Here $E_P(v_i, v_j)$ is the attribute vector for the edge (v_i, v_j) in G_P between pattern vertices V_i and V_j. Similarly $E_D(f(v_i), f(v_j))$ is the attribute vector for the corresponding edge $(f(v_i), f(v_j))$ in G_D. Further, $dis(A_{V_P}(v_i), A_{V_D}(f(v_i)))$ and $dis(E_P(v_i, v_j), E_D(f(v_i), f(v_j)))$ denotes the inexactness of matching the feature vectors for vertices and edges respectively. This distance function, generally stated as $dis(Q(x),\ Q(f(x)))$ where *Q* is the term for which the vector is being measured, defines the inexactness in the matching process.

The ideal case is when $E_P(v_i,\ v_j) = E_D(f(v_i),\ f(v_j))$ and $A_{V_P}(v_i) = A_{V_D}(f(v_i))$ i.e, the distance functions for vertices (V_{dis}) and edges (E_{dis}) are zero. So, it can be said that in absence of the distance function inexact matching becomes exact matching. In other words, exact matching is a special case of inexact matching where the distance function is zero.

Our aim is to select an *f* in such a way so that it minimizes the aggregate value of $V_{dis} + E_{dis}$. If a weighted aggregate is used then the expression may be denoted as $\alpha.E_{dis} + (1-\alpha).V_{dis}$, where α is the weight of E_{dis} in the aggregate and $0<\alpha<1$. Since minimizing the aggregate value is in general intractable (an NP-hard problem) and it is not possible to find out if the value has really been minimized. Hence we use greedy method to select f that makes the distance measure small but not necessarily minimal. It is to be noted that mathematically the notation $f(V_{P_i})$ actually denotes *f(i)* and may be used sometimes interchangeably.

SECONDARY GOAL

There may be circumstances where there may not be any prior information about any criminal group. Under such circumstances, based on some predetermined conditions we can extract information from the present database to detect the prime potential suspects. Also using the basic terms of Social Network Analysis we can find people affiliated to those suspects or in other terms can detect a possible criminal group.

Another goal may be to detect the role of the individuals in the detected network. Using the attribute values of an individual or a node and also by detecting its betweenness and

closeness values, we can determine the role of an individual. This information is often very useful in understanding the information flow and the workflow in a criminal network.

To prevent criminal or terrorist activities law enforcement agencies try to track down such individuals, who play key roles in the operation of the network. We can find network efficiency for the suspects and determine the most efficient ways to disrupt the network to affect its workflow.

In a practical scenario, analyzing the mentioned problems will give a complete view of a network.

2.5 Pre-processing of the Relational Data

The data used for these experiments were collected from Dr. Sofus Macskassy, Dr. Sandeep Maripuri, and Dr. Eric Rickard and are simulated based on real terrorist activities. So, the representation of the problem and the extraction of important data from the pool of available data was a challenging task. Also it is quite evident that if the data is noisy, finding a match for a pattern can be difficult. Since we can have this kind of scenario very often in practical setups we chose to intentionally to insert perturb our data accordingly to simulate a similar situation. It has many tables describing how the individuals in a social network carried out certain events. Due to the complex nature of the data we tried to limit the length of the feature vector and focus on the algorithm rather than to get involved with the computational difficulties.

The terrorist database is a relational database consisting of 87 tables, containing information about 300 individuals, their participation (and roles) in various events, including exploitation events and communication events (as two-way and n-way communications). The individuals are shown in the database as either a threat person (a terrorist) or non-threat person. So one could define features like “the number of times communicating to a threat person” for a vertex. Due to the complex nature of the data we tried to limit the length of the feature vectors for vertex and for edge. Through a careful analysis of the database, we have selected 17 attributes for vertices and 7 attributes for edges.

So, an immediate question that comes to mind is, why have we selected only those 300 individuals from a dataset consisting of records for more individuals?

We observed the attribute values of all the individuals and noticed that some attributes have distinctively larger value than others. So we created a threshold based on average value of the attributes and excluded individuals that did not meet the threshold values for those attributes. Since we also run many folds of the data graph we eventually test all sizes of data graph irrespective of the initial size. Due to memory block size restriction in C compiler we can not test more than 7 folds. Similarly, if the initial data graph was bigger it could only be folded only n times where n is less than 7 and is dependent on the initial data graph size and the memory block limitation size of C compiler.

Another question that might follow is that, what are the criteria to select those 17 attributes for vertices and 7 attributes for edges?

Although we keep most of the attributes to characterize the vertices and edges, we exclude some of them to avoid unnecessary calculation. We keep the all the exploitation events since they are one of the most useful attributes to detect terrorists. From the rest of the attributes, if the average value for the attribute is very close to zero we exclude that attribute. The only exception is the evidence attribute, which, by nature, may come very rarely but might be important to detect a potential terrorist.

Some Definitions:

Not everyone in the terrorist group work in the same way. Some act from behind-the-scene and master mind the process whereas some others go to the target scene and carry out the attacks. In the following events we have seen that in any event there are two distinctive groups among the individuals involved in the events.

1. **Directing Agent** or **Initiator**: These people usually direct the events or in many cases initiate the events. Intuitively these are the people closer to the terrorist network hierarchy. The term “Initiator” is used in some of the communication events and the purpose of this individual is to initiate a communication process that will facilitate the goal.

2. **Deliberate Actors** or **Respondent**: These are they people who usually carry out the task on site and follow a directing agent’s instructions. Intuitively these people are usually in the lower tiers of terrorist network hierarchy. The term “Respondent” is used in some of the communication events to refer people who respond to and initiator and participate in a communication event.

An individual may sometimes act as a directing agent as well as a deliberate actor or an initiator as well as a respondent.

The individuals and their relations were characterized based on these seven primarily chosen events, which we thought would represent important aspects of terrorist activities:

A. Exploitation Events: Since exploitation of individuals, circumstances and resources are one of the most important features of a secretive affair like a terrorist attack, exploitation events were chosen to be an integral part of the test database.

1. Exploitation Consummation Event:

This event shows people who completed or consummated an exploitation process that may be useful for a future terrorist attack. For this event we focus on the tables `exploitationconsummationevent` and `exploitationconsummationevent_deliberateactors`. The first one lists the individuals involved in such events as “directing agents” or the people who guides the event and the second list consists of individuals termed as “deliberate actors” or the people, who participates in these events deliberately and works mainly as performers not guide.

2. Productivity Exploitation Event:

This event shows people who exploited his own or some other individual’s productivity to make it useful for a future terrorist attack. For this event we focus on the tables `productivityexploitationcase` and `productivityexploitationcase_deliberateactors`. Just like the previous event, in this case also the tables contains records of directing agents and deliberate actors respectively involved in all productivity exploitation cases.

3. Vulnerability Exploitation Event:

This event shows people who exploited some individual’s vulnerability or vulnerability of a situation to gain some advantages that might be useful for a future terrorist attack. For this event we focus on the tables `vulnerabilityexploitationcase` and `vulnerabilityexploitationcase_deliberateactors`. Just like the other exploitation events, in this case also the tables contains records of directing agents and deliberate actors respectively involved in all vulnerability exploitation cases.

B. Communication Events: Communication is always a very important part of any social network. It is even more important in a terrorist network because the members of a terrorist group, who have been assigned a task, need to be in touch with each other to convey different messages.

4. Chained Communication Event:

In this event a directing agent initiates the process and a message reaches a destination using other individuals as intermediate transmitters. Other than the directing agent, all the other individuals involved in this event are deliberate actors. In this process every individual does not necessarily communicate directly with all the other individuals but the same message is conveyed to the whole group. The tables used to acquire the data are `chainedcommunicationevent` and `chainedcommunicationevent_deliberateactors`. We get the directing agent and deliberate actor information from these tables respectively.

5. n-way Communication Event:

This event is a group discussion and is very similar to a conference call where everyone can speak to everyone at the same time. When there is a need of such a communication an initiator initiates the process and other affiliated individuals, who are called respondents, respond to that call. The tables used to acquire the data are `nwaycommunicationevent` and `nwaycommunicationevent_iterespondent`. We get the initiator and respondent information from these tables respectively.

6. Two-way Communication Event:

This event shows how many times two individuals have communicated only among themselves. The caller can be seen as the initiator and the receiver can be seen as a respondent. Any individual can be initiator and respondent in two different cases. Two-way communication information was obtained from the table `twowaycommunicationevent`, where both initiator and respondent information is available.

C. Miscellaneous: There could be many other events involving resources (acquiring them and using them), capabilities of the individuals, studying the targets (visiting or observing). Among these we have selected three based on the type of data we have:

7. Observing Targets Event: Observing a target for many times by a terrorist individual is a common practice and hence can be useful for our database. The information on individuals observing targets can be found on `observing_deliberateactors`. Information like, which individual observed a target and how many times, can be obtained from this table.

8. Evidence: While executing the plan, terrorists may leave evidences in form of eye witnesses and other forensic traces. This information may be useful to identify a terrorist suspect. We have counted the number of evidences for each individual from the tables `applycapabilityevent` and `applyresourceevent`.

9. Group Status: Group status of an individual defines whether that individual is listed in a threat group or a non-threat group. This information is available in the tables `nonthreatgroup_memberagents` and `threatgroup_memberagents` and can be availed accordingly.

Table 1. First part of the table of attribute values for the Pattern graph vertices

	Exploitation Consummation		Productivity Exploitation		Vulnerability Exploitation		Observing Targets		
Individuals	Agent	Actor	Agent	Actor	Agent	Actor	Actor	# of Targets	# of Times
In-10376	0	0	0	0	0	0	0	1	03
In-13380	0	0	0	0	0	0	0	2	10
In-14003	0	0	0	0	0	4	2	2	11
In-15606	0	0	0	0	0	1	1	2	08
In-2131	0	0	0	0	0	1	0	1	02
In-22983	0	0	0	0	0	0	0	2	05
In-24100	0	0	0	0	0	0	0	2	16
In-24164	0	0	0	0	0	0	0	2	11
In-4433	0	0	0	0	0	4	4	2	13
In-608	0	0	0	0	0	0	0	2	05
In-850	0	0	0	0	0	0	0	2	04
In-8780	0	0	0	0	0	2	1	2	05

For individuals or the nodes, in most of the events there are two tables to extract the data. So, we mainly approach the tables to find out how many times an individual has acted as directing agents or initiators (from the tables with directing agent or initiator information in them) and how many times as deliberate actors or respondents (from the tables with deliberate actor or respondent information in them). This information enables us to see who plays what kind of role and to what extinct.

Table 2. Second part of the table of attribute values for the Pattern graph vertices

	Chained Communication		n-way Communication		Two-way Communication		Group	Evidence
Individuals	Agent	Actor	Initiator	Respondent	Initiator	Respondent	Status	Present
In-10376	0	0	1	06	16	20	0	0
In-13380	0	0	1	06	14	11	0	0
In-14003	0	1	1	05	27	26	1	1
In-15606	0	2	1	03	13	12	1	1
In-2131	0	0	1	03	13	21	0	0
In-22983	0	0	1	10	17	22	0	0
In-24100	0	0	5	08	35	27	0	0
In-24164	0	0	2	06	29	19	0	0
In-4433	0	0	2	07	37	30	1	0
In-608	0	0	0	08	17	17	0	0
In-850	0	0	1	08	19	29	0	0
In-8780	0	0	2	06	11	14	1	0

Table 3. A snippet of the table of attribute values for the Data graph relationships

Individual	Individual	Exploitation Consumm.	Vulnerability Exploitation	Productivity Exploitation	Observing Targets	Chained Comm.	n-way Comm.	2-way Comm.
In-10004	In-27082	0	0	0	1	0	5	00
In-10004	In-6318	0	0	0	1	0	6	02
In-10103	In-2462	0	0	0	1	0	0	00
In-10126	In-1280	1	0	0	2	1	0	00
In-10126	In-13286	0	0	0	0	0	0	06
In-10126	In-19219	0	0	0	0	2	0	08
In-10126	In-27771	0	1	1	2	3	0	00
In-10126	In-27828	1	0	0	2	1	0	00
In-10126	In-7256	0	0	0	0	0	0	06
In-10126	In-9929	0	1	1	3	3	0	07
In-1014	In-10243	0	0	0	0	1	0	00
In-1014	In-15606	0	0	0	0	1	0	00
In-1014	In-22972	0	0	0	1	0	0	00
In-1014	In-2556	0	0	0	2	0	0	02
In-1014	In-27082	0	0	0	2	0	0	00
In-1014	In-28675	0	0	0	0	1	0	00
In-1014	In-4026	1	0	0	0	0	2	00
In-1018	In-2742	0	0	0	0	0	0	03
In-1018	In-9686	0	0	0	0	0	0	03

For relations or the edges, we have created views by merging the directing agent/initiator table and deliberate actor/respondent tables, to find out a complete list of participants in that event. Then we have created all possible combinations by joining the view to itself and

measuring how many times each relation exists. Here the relationships are directed. Say, if individual P is related to individual Q then relation PQ and QP both exist in the database, although both of them have same feature vector. The relations between two individuals are not seen on the basis of their role. It counts how many times an individual have been involved with another individual to carry out similar events. From the tables above one can get an idea about the attributed pattern graph, which is basically a subset of the data graph. Similarly the data graph is a 300 node graph with each node having values for the same attributes.

2.6 Similarity Measures Used in Algorithms

Say two graphs, a pattern graph $G_P = (V_P, A_{V_P}, E_P)$ and a data graph $G_D = (V_D, A_{V_D}, E_D)$, are used to represent the problem. We assume that $|V_P| \leq |V_D|$, which is the case when a match for a smaller network is tried to found from a bigger social network.

One solution to this non-bijective graph matching problem can be defined by finding a set of association between the pattern graph and the data graph. For vertex association we have to make sure that each node of G_P is associated with at least one node of G_D. Edge associations can be derived from the vertex associations where $(v_i,\ v_j) \in E_P$ is associated with all $(v_i',\ v_j') \in E_D$, where either v_i is associated to v_i' and v_j is associated to v_j', or v_i is associated to v_j' and v_j is associated to v_i'.

An effective solution in such a case depends on high similarity values of the associations. There can be two similarity entities: vertex similarity and edge similarity. The vertex similarity depicts similarity between two individuals and the edge similarity depicts the similarity between relationships among individuals from one graph to another. We have used different types of similarities to find out how differently the algorithm behaves when different similarities are used. All the similarities we have used are normalized and are stated as following:

1. All Position Match:

In this kind of similarity matching we try to see if the value for *i*-th feature of $v_i \in V_P$, matches with the value for *i*-th feature of $v_i' \in V_D$ and *i* consists of values from 1 to *p* (as stated in the Problem Statement; Section 2.4). This is useful when all the attributes are equally significant in determining a match. The value is normalized by dividing the

aggregate of total number of matches by total number of matches possible, i.e., total number of features for a vertex (*p*). This similarity can be written as,

$$Sim_{APM}(v_i, v_i') = \frac{1}{p}\sum_{i=1}^{p} x_i ,$$

where x_i =1, when the value in the i-th position of v_i and v_i' matches, x_i =0 if otherwise.

2. Positive Position Match:

In this kind of similarity matches the value for *i*-th feature of $v_i \in V_P$, matches with the value for *i*-th feature of $v_i' \in V_D$, where the value for *i*-th feature of v_i and v_i' are non-zero. Significance of this similarity lies in the fact that not all the features may be useful all the time. So if the value of any feature is zero, it means that the particular characteristic might be absent in the individual or a particular kind of relation may not be present between two individuals. For normalization purpose the aggregate of total number of matches is divided by total number of features for which both v_i and v_i' have positive values (say *k*). This similarity can be written as,

$$Sim_{PPM}(v_i, v_i') = \frac{1}{k}\sum_{i=1}^{k} x_i ,$$

where x_i =1, when the value in the i-th position of v_i and v_i' are both positive and matches, x_i =0 if otherwise.

3. All Position Weighted Match:

In this kind of similarity while matching $v_i \in V_P$ to $v_i' \in V_D$, we assign less credit to features with zero value for either v_i or v_i' or both. Only those features which have positive values for both v_i or v_i' are assigned more credit. The weighted system automatically calculates the credits for zero and non-zero features and the aggregated credit is normalized by dividing it with the total number of matches possible, i.e., total number of features for a vertex (*p*). Say there are *a* number of features for which both v_i or v_i' have some positive value and a' of them have matches, then the similarity can be written as,

$$Sim_{APWM}(v_i, v_i') = \frac{1}{p}\sum_{i=1}^{p} x_i ,$$

where $x_i = \frac{1}{4(p-a)}$ when *i*-th feature vector for either v_i or v_i' is zero,

x_i =0 when *i*-th feature vector for both v_i or v_i' is positive and does not match,

$x_i = \frac{3}{4a}$ when *i*-th feature vector for both v_i or v_i' is positive and matches.

4. Cosine Match:

Measuring similarity for both vertices and the edges is an integral part of the algorithms. We have also used cosine similarity as one of our similarity measures and has used it uniformly through out all the algorithms because we noticed that this type of similarity match always generated more realistic similarity scores, once normalized .

For each pair of vectors V_i and V_j, where $V_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ and $V_j = (v_{j1}, v_{j2}, \ldots, v_{in})$, the similarity between V_i and V_j can be given by, $Cosine = \frac{\sum_{k=1}^{n} v_{ik} . v_{jk}}{\sqrt{\sum_{k=1}^{n} v_{ik}^2} . \sqrt{\sum_{k=1}^{n} v_{jk}^2}}$.

In an alternative method, the vectors could be normalized in the first place so that the dot product between values for the same attributes of two vectors produce the similarity value directly. This saves the calculation of the root mean square every time we try to access a vector to find a similarity measure.

3 Proposed Models

In this part we present theoretical models to address the problem of approximate graph matching.

3.1 Weighted Incremental Greedy Model

In this method we apply an incremental algorithm to make a greedy search in which the best mapping of the pattern nodes to the data nodes is calculated with help of an objective function.

Our goal is to find a mapping *f*, such that $f: V_P \rightarrow V_D \cup \{NULL\}$, which could also be used as partial mapping during the intermediate stages of the algorithm [23]. In lieu with the $V_{dis}(f)$ and $E_{dis}(f)$ functions, we can define vertex similarity and edge similarity functions as,

$$sim_v(f) = \frac{1}{|V_P|} \sum_{\substack{v_i \in V_P \\ f(v_i) \neq NULL}} sim^*[A_{V_P}(v_i),\ A_{V_D}(f(v_i))] \text{ , and}$$

$$sim_e(f) = \frac{1}{Edge\ Count(V_P)} \sum_{\substack{(v_i, v_j) \in (V_P \times V_P) \\ f(v_i) \neq NULL \\ f(v_j) \neq NULL}} sim^*[E_P(v_i, v_j),\ E_D(f(v_i), f(v_j))]$$

These $sim_v(f)$ and $sim_e(f)$ functions, lead to our objective function *SIM* (*f*), when subjected to a weighted accumulation. Hence the objective function can be stated as,

$$SIM(f) = \alpha.\ sim_v(f) + (1-\alpha).\ sim_e(f)$$

With this objective function, first we try to find one set of match of the pattern graph inside the data graph which is stored in an array $M(n)$ where *n*= total number of vertices in pattern graph. This algorithm (say A_1) can later be used in an iterative way to find all possible matches in the data graph.

The algorithm A_1 can be described as follows:

Algorithm A1: Weighted Incremental Greedy

Input:

1. An attributed data graph $G_D = (V_D, A_{V_D}, E_D)$ has to be provided which will act as a social network structure.
2. An attributed pattern graph $G_P = (V_P, A_{V_P}, E_P)$ has to be provided which will act as a model terrorist group.

Output:

A one dimensional array $M(n)$ where *n*= total number of vertices in pattern graph and function *M* denotes a match of a pattern vertex found in the data vertices.

Algorithm:

(1) initialization of *f*: map all pattern vertices to *NULL*

(2) **while** there exists v_i in V_P mapped to *NULL* **do**

for each pair of vertices $(v_i, v_j) \in (V_P \times V_D)$ such that $f(v_i) = NULL$ and $f^{-1}(v_j) = NULL$

begin

$f_{ij} \leftarrow (f - \{(i, NULL)\}) \cup \{(i, j)\}$;

compute $SIM(f_{ij})$;

end

end_for;

Select one $f_{i^*j^*}$ with maximum $SIM(f_{ij})$;

update $f \leftarrow f_{i^*j^*}$;

end_while;

The definition of f_{ij} has been presented in such a way so that it could comply with the intermediate steps with partial mappings. As we define,

$$f: \ V_P \rightarrow V_D \cup \{NULL\}$$
$$f^{-1}: \ V_D \rightarrow V_P \cup \{NULL\}$$

from the representation of f in Figure 3. where pattern vertices x and y have been mapped to data vertices a and b respectively, we can say,

$$f(x) = a, \ \ f(y) = b, \ \ f(z) = NULL$$
$$f^{-1}(a) = x, \ \ f^{-1}(b) = y, \ \ f^{-1}(c) = f^{-1}(d) = f^{-1}(e) = NULL$$

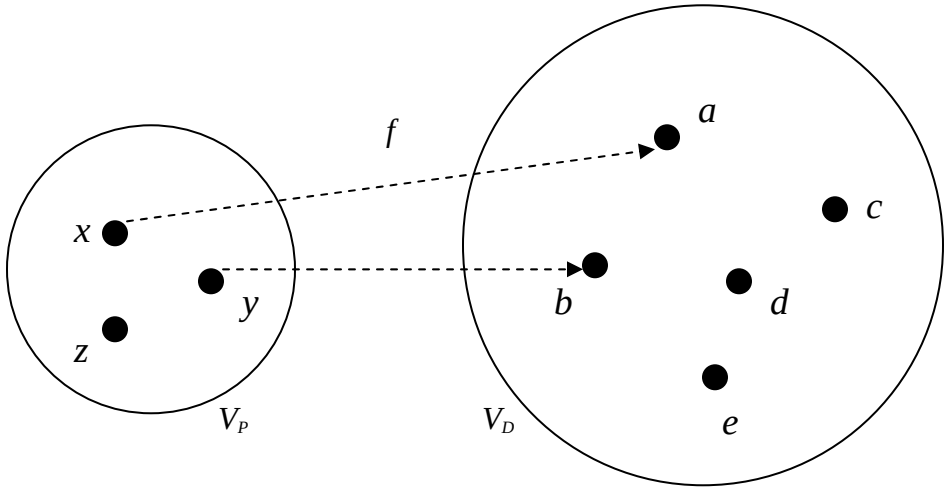

Figure 3. Mapping of f and f^{-1} in greedy matching

If the size of the pattern graph is V_P and the size of the data graph is V_D, the complexity of the greedy method can be denoted by $O(V_P^2.V_D)$. The each pattern vertex is being searched for in all the data vertices so that gives rise to a complexity of $V_P \times V_D$. This process has to be done to get match for all pattern vertices, so the complexity increases V_P times and results in $O(V_P^2.V_D)$. A more accurate calculation can be done keeping in mind that in each step the search space decreases. In that case the complexity can be written as $\sum_{i=0}^{V_P-1}(V_P - i)(V_D - i)$. It can be showed by approximation, it produces the same time complexity as $O(V_P^2.V_D)$.

3.2 Degree-based Incremental Greedy Model

In this method we have used the degree information from the graphs to reduce the search space. Since a vertex can be characterized by its degree, it is necessary for a pattern vertex that its degree must be greater than the degree of the data vertex it is being matched with.

In this way we can reduce the search space and hence avoid the similarity calculation for cases, which satisfy these criteria. So the algorithm changes in the following way whereas the output and input remains the same.

Algorithm A2: Degree-based Incremental Greedy

Input:

1. An attributed data graph $G_D = (V_D, A_{V_D}, E_D)$ has to be provided which will act as a social network structure.
2. An attributed pattern graph $G_P = (V_P, A_{V_P}, E_P)$ has to be provided which will act as a model terrorist group.

Output:

A one dimensional array $M(n)$ where *n*= total number of vertices in pattern graph and function *M* denotes a match of a pattern vertex found in the data vertices.

Algorithm:

(1) initialization of *f*: map all pattern vertices to *NULL*

(2) **while** there exists v_i in V_P mapped to *NULL* **do**

for each pair of vertices $(v_i, v_j) \in (V_P \times V_D)$ such that $f(v_i) = NULL$ and $f^{-1}(v_j) = NULL$

if Degree**(** v_i **)** is less than or equal to Degree**(** v_j **)** **then**

$f_{ij} \leftarrow (f - \{(i, NULL)\}) \cup \{(i, j)\}$;

compute $SIM(f_{ij})$;

end_if;

end_for;
Select one $f_{i^*j^*}$ with maximum $SIM(f_{ij})$;
update $f \leftarrow f_{i^*j^*}$;
end_while;

Although we save the similarity calculations for some of the cases, we still need to review each case to test the criteria and hence spend the time needed for the comparisons.

3.3 Neighbors' Degree-based Incremental Greedy Model

This model follows subsequently from the degree-based incremental greedy model. Here we reduce the search space even more by screening them based on the characteristics of their neighbor. So, for each node we select its neighbor with the most number of connections and while matching from pattern graph vertex v_i to data graph vertex v_j, we test for the condition that the neighbor of v_i that has the largest degree among all of its neighbors should have a lesser degree value than the neighbor of v_j that has the maximum degree among its neighbors. With this criterion we reduce the search space even more and the more connected the pattern graph is theoretically it should provide more accurate results.

Algorithm A3: Neighbors' Degree-based Incremental Greedy

Input:

---

1. An attributed data graph $G_D = (V_D, A_{V_D}, E_D)$ has to be provided which will act as a social network structure.
2. An attributed pattern graph $G_P = (V_P, A_{V_P}, E_P)$ has to be provided which will act as a model terrorist group.

Output:

A one dimensional array $M(n)$ where *n*= total number of vertices in pattern graph and function *M* denotes a match of a pattern vertex found in the data vertices.

Algorithm:

---

(1) initialization of *f*: map all pattern vertices to *NULL*

(2) **while** there exists v_i in V_P mapped to *NULL* **do**

for each pair of vertices $(v_i, v_j) \in (V_P \times V_D)$ such that $f(v_i) = NULL$ and $f^{-1}(v_j) = NULL$

if Degree**(** v_i **)** is less than or equal to Degree**(** v_j **)** **AND** NeighborDegree**(** v_i **)** is less than or equal to NeighborDegree**(** v_j **)** **then**

$f_{ij} \leftarrow (f - \{(i, NULL)\}) \cup \{(i, j)\}$;

compute $SIM(f_{ij})$;

end_if;

end_for;

Select one $f_{i^*j^*}$ with maximum $SIM(f_{ij})$;

update $f \leftarrow f_{i^*j^*}$;

end_while;

It is to be noted that in our graph representation, the notion of neighbor of a vertex *v* in G is defined as the vertices in G to which vertex *v* is connected with at least one (but not necessarily all) non-zero attribute value on the edge. Here, NeighborDegree**(** v_i **)** is the degree of the neighbor of v_i that has maximum degree among all the neighbors of v_i.

3.4 Important Features of the Greedy Model

In this section we will be discussing two important features of the greedy model we have implemented.

3.4.1 Introduction of Noise by Perturbation

A social graph is always a dynamic one. For easier calculations we assume that the number of vertices and edges remain same every time. But depending on various circumstances in a social network it is very likely that attributes for each vertex or edge might change. To implement that change we intentionally insert some noise.

Amount of perturbation basically denotes the amount of noise intentionally inserted into the vertex and edge vectors for the data graph vertices, which are target copies of pattern vertices. The attribute values of the vertices and the edges range anywhere from 0 to 25. Although for most of the attributes, values are limited to 10. So we have tested the algorithms for perturbations= 0, 1, 2, 3 where perturbation is equal to *i* means that every

time an attribute value is picked to be changed, a random value ranging from 0 to *i* is chosen and the attribute value is changed (plus or minus) by that amount. The change amount is not stored and the next attribute value is changed by another randomly chosen amount.

Another way to insert the noise may be based on the average value of each attribute. Since changes in unitary standard are comparable to our data we can change the attribute the data by 0, 1, 2, 3 etc and count it as noise. But if the attribute values are not comparable to unitary decimal numbers then we could take average value of each attribute for all the vertices or all the edges as a unit and insert that unit or its multiple as noise. In general cases this would be a more logical approach to make the perturbation. In our program we had the option of making the perturbation based on the average value of the attributes.

3.4.2 Notion of a Seed

A seed is a starting point of a process that is chosen to facilitate the goal of the process. Here, in context to our problem it can be defined as a smart initial point to initiate the greedy method. Since we aim to reduce run-time with the help of degree information we also need to investigate the accuracy more closely. Accuracy is not affected if there is no noise or perturbation in the data. But with high amount of noise accuracy becomes a problem.

So we consider a practical situation where there is high amount of noise in data or to say some of the attributes have gone through some apparent change. In such a case, it is possible that a highly accurate prediction of a possible terrorist group is more important than finding it in a matter of seconds. So under these circumstances one could think of a more accurate starting point by sacrificing some run-time and hence increasing the accuracy. Seed does not necessarily always increase accuracy and decrease efficiency. The balance of accuracy and efficiency is maintained by the method that is used to compute the seed. There can be cases were seeds are chosen in a smart way to increase the efficiency. But in our case the method that calculates the seed, increases the accuracy by sacrificing the efficiency.

The purpose of the seed is to provide the greedy algorithms with a useful head-start[24]. In many cases it is possible that incremental greedy method chooses its first match wrongly. If the first match is wrong, since the later vertices added through incremental greedy

calculates the similarity with that initial erroneous mapping, and the error in finding the next vertex keeps accumulating. So, in order to get a good head-start, we will first map a pair of vertices (and the edge between them) from the pattern graph to a pair of data graph vertices based on an exhaustive search and start our incremental greedy searches from that point onwards. We have used a two vertex seed in some of our experiments. More the number of vertices and edges involved in the exhaustive search, the seed works better but the run-time becomes considerably higher. The degree information can also be used in finding the seed and make the seed finding process faster. Hence this kind of seed can be called a degree-based seed.

3.5 Greedy Model with Partial Pattern and Post-matching Expansion

In some practical scenarios, it is possible that knowledge about a part of the target graph is available. So in that case, our goal should be to identify that partial graph from the data graph and starting from that matched sub-graph in data, a string of processes are executed that gives rise to the expansion of the current match and find strongly connected elements or potential terrorists.

- The first step in this method is the partial sub-graph matching. Once the target sub-graph is found in the data, they are tagged and used as an initial point. Any of the greedy algorithms can be used for the partial sub-graph matching.

- In the next step, attributes of the current match are analyzed. Based on the matched nodes the attributes of the values of the vertices and the edges are analyzed. The attributes for both vertices and edges are ranked and only the top one-third is selected for the post-matching expansion. The goal of this step is to emphasize the most important attributes and ignore the less important attributes.

- In the next step, we create a function called “strength”. Say, a vertex has been chosen to be tested for a strong connection to the current match. So its “vertex strength” would be the sum of the values of the top one-third selected vertex attributes. Then it would be tested for its edges. It will similarly accumulate “edge strength” for all the possible edges with the nodes of the current match. For each edge the “edge strength” would be the summation of the values in the top one-third selected edge attributes. So for each vertex, “total strength”, which is the sum of “vertex strength” and “edge strength”, is calculated.

- The one with the biggest “total strength” is considered to be the most strongly connected. This node is then included in the current match and the whole process is repeated.
- This process can be repeated for a desired number of times to add desired number of nodes based on their strength of connection.

4 Experiments and Results

We have written a C++ program for the experimental study, and executed it on a HP desktop with 2.13 GHz Intel Core Duo microprocessor and 2 GB RAM memory, running Windows XP/SP2. Several experiments have been performed to compare the performance of the incremental greedy method vs. the degree-based incremental greedy method, using pattern and data graphs extracted from the synthetic terrorist database described in the previous section. To accommodate noisy data, experiments have also been conducted to see the behavior of the algorithms when the original data is perturbed by various amounts.

4.1 Experimental Setup

Under practical circumstances the size of the pattern graph could be from 10 to 20. A bigger pattern graph can also be divided into smaller graphs and can be treated as a separate smaller problem. We have used our pattern graph size to be 12 and data graph size to be 300 based on the data. We have increased the data graph size up to seven folds (which means a data graph of around 2000 vertices) to test whether detection of multiple copies of the pattern graph is possible or not.

In the simplest case with a pattern graph of size 10 to 12, and a data graph of size 300, which contains sub-graph that is a copy or an approximate copy of the pattern graph, we ran both the incremental greedy and degree-based incremental greedy algorithms and compare them from two aspects: computational efficiency and matching accuracy. The efficiency is measured by computational time (in seconds) and the accuracy is measured by the percentage of correct vertex mappings (since we know the correct target sub-graph in the data graph). Of course the matching accuracy could be alternatively measured by the similarity scores achieved by the mapping function f. In experiments with data graph size increased to be k-folds of the original data graph, we essentially made k copies of the original data graph, and ran each greedy algorithm k times, in search for k copies of the pattern graph.

Since we do not have availability to a larger dataset or even a different dataset we have increased the dataset by multiple folds to see how the algorithm acts while finding a match from a bigger dataset. While connecting the folds of the data graph we have tried different options to see how increase in connection links affects the performance of the algorithm. We have tried the same algorithms for three different cases in terms of existing number of

connections between each pair of folds. For the first case there is no connection between a pair of folds. For cases 2 and 3 there are 30 and 60 respectively. It can be argued about what is an optimal number of connections for a graph to be increased by multiple folds. But since the number of edges in our data graph is 634, we go by one-twentieth and one-tenth of the number of edges. So, we basically introduce around 30 or 60 randomly generated edges between all combinations of the copies of the dataset. Say, if the dataset is folded 3 times, there are approximately 60 edges present as connection between fold 1 and fold 2, another 0 between fold 1 and fold 3 and another 30 between fold 2 and fold 3. Deciding the number of connections can be subject to arguments and there are no absolute criteria to decide it.

As an example, if the data is folded 7 times with 60 connections between each pair of folds, number of edges inside the folds equals to $7\times$ 634 i.e., 4438 and total number of connection between all the folds equals to ${}^{7}C_{2} \times 60$ i.e., 2520. So the total number of edges in the resultant data graph would be 6958.

4.2 Results

Figure 4 and Table 4 shows the graph and its data for the computational time comparison between incremental greedy (IG), degree-based greedy algorithm (DIG) and neighbors' degree-based greedy algorithm (NDIG). Run-time is independent of the level of perturbation, whereas accuracy is dependent on perturbation.

Table 4. Comparing run-times of IG, DIG and NDIG methods using a dense pattern graph for different fold connections and different seeds.

dvn= 284 * k	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6	k = 7
(all units in secs)	dvn = 284	dvn = 568	dvn = 852	dvn = 1136	dvn = 1420	dvn = 1704	dvn = 1988
pvn=12	no seed						
No connection							
IG:	0.006	0.0235	0.052	0.0777	0.125	0.174	0.227
DIG:	0.0035	0.016	0.031	0.0622	0.0782	0.1018	0.149
NDIG:	0.002	0.0075	0.019	0.345	0.0588	0.0735	0.0932
Connections= 30							
IG:	0.006	0.0235	0.052	0.078	0.128	0.175	0.228
DIG:	0.0035	0.015	0.0366	0.0625	0.0782	0.123	0.154
NDIG:	0.002	0.0085	0.0187	0.387	0.0596	0.0731	0.0954

Connections=60							
IG:	0.006	0.0235	0.0545	0.0785	0.128	0.177	0.225
DIG:	0.004	0.016	0.0363	0.0625	0.0902	0.138	0.1854
NDIG:	0.0025	0.009	0.0203	0.0392	0.0635	0.088	0.1028
pvn=12	two-vertex seed						
IG:	2.813	22.078	74.062	174.983	342.297	588.484	934.42
DIG:	2.813	22.062	74.063	175.03	341.89	587.187	931.297

(table continued from previous page).

Although we have tested these three algorithms for different pattern sizes and with different number of connection folds, we use the data with pvn=12 to draw the graph seen in Figure 4 and 5. Given a pattern graph size (denoted by the number of the vertices) our program randomly selects a pattern graph (from the data graph) of that size and tries to find a match of it from the data graph by executing the algorithms. Pattern graphs with different sizes were tested to observe whether the trend of the experimental result is similar for all the different sizes. In other words, similar results for randomly picked pattern graphs of different sizes substantiate the results.

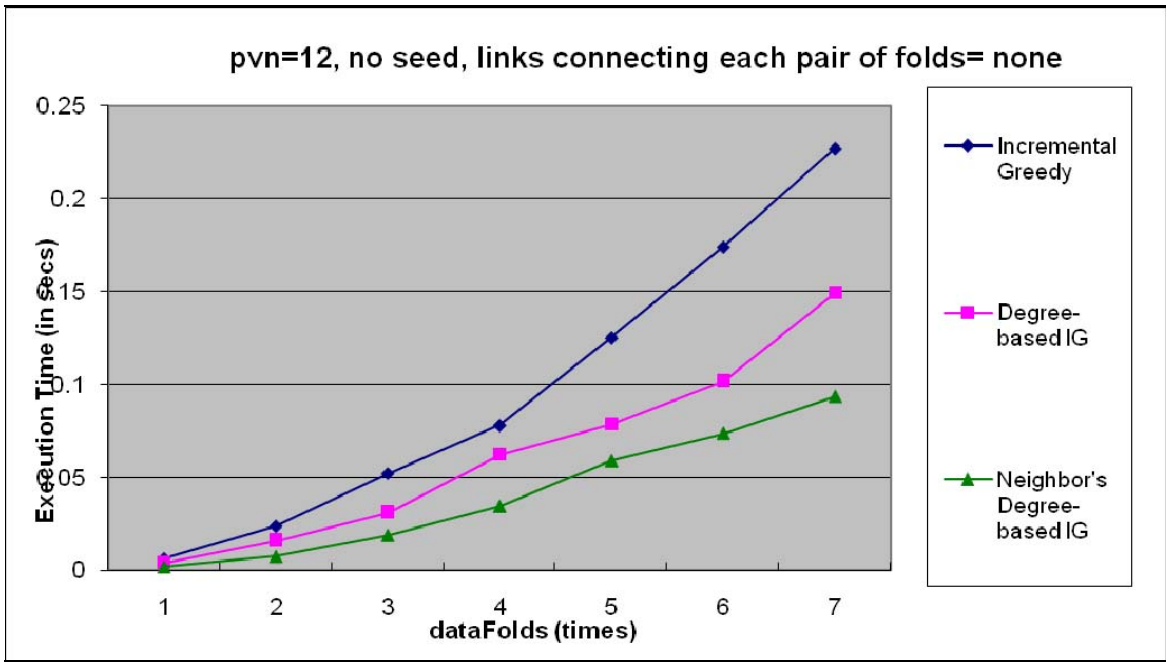

Figure 4. Comparing performances of original and Degree-based Greedy algorithms (with no seed, no fold connection and pvn=12 for a dense pattern graph)

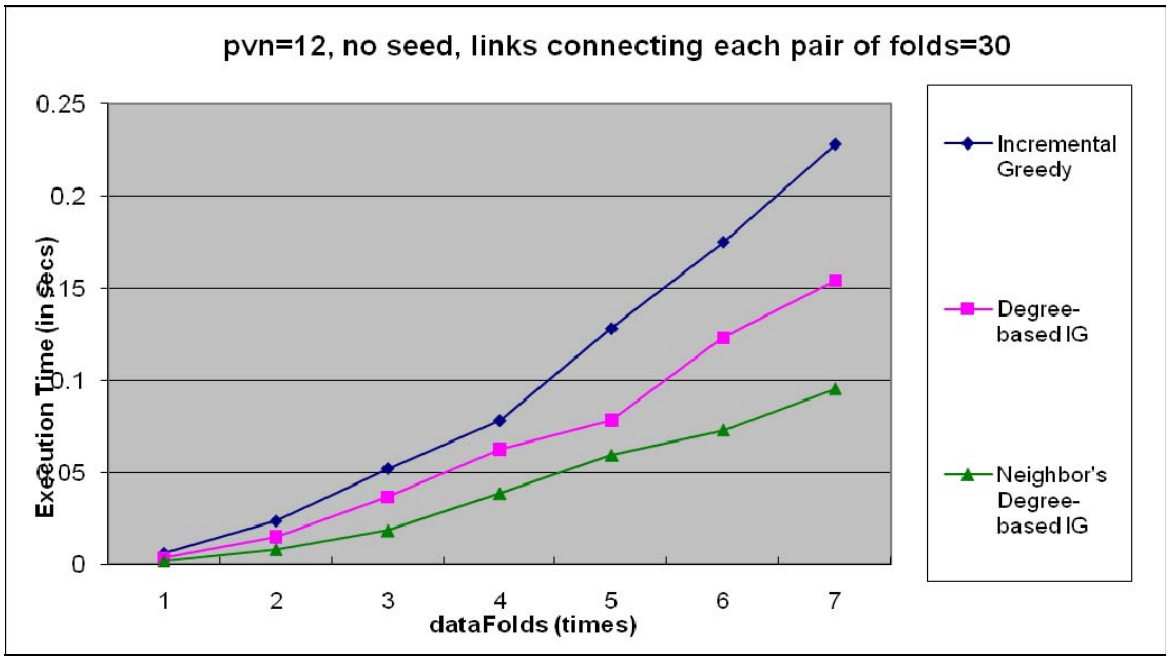

Figure 5. Comparing performances of original and Degree-based Greedy algorithms (with no seed, 30 fold connections and pvn=12 for a dense pattern graph)

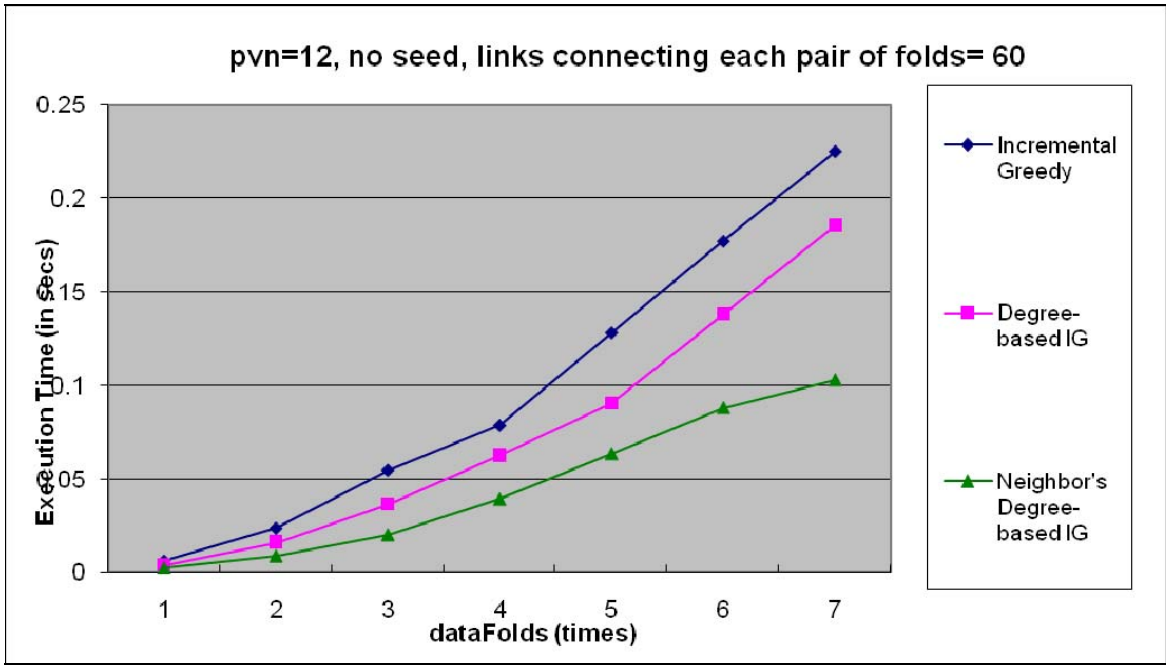

Figure 6. Comparing performances of original and Degree-based Greedy algorithms (with no seed, 60 fold connections and pvn=12 for a dense pattern graph).

It is evident from the experiment results that for every different pattern vertex sizes degree-based incremental greedy works faster than the original incremental greedy and using the neighbors' degree information makes the algorithm even faster. When run without any noise the algorithms show 100 percent accuracy as expected. Even with perturbation level 1, accuracy is around 100 percent for most of the cases. The results show that there is not a noticeable variation in the run-time with increase in number of connecting links between pair of folds.

There is a noticeable difference between the run-times with a dense pattern graph and a sparse pattern graph. We have tested the sub-graph matching problem with a selected pattern graph that is dense and also with randomly generated pattern graphs, which are presumably sparse. From theory it is clear that our algorithm is dependent on number of edges and hence the degree of the vertices of the pattern graph to show a performance improvement for degree-based and neighbors' degree-based greedy methods. If the pattern graph has more edges and pattern vertices have more degree then we can skip many data vertices while testing for the degree requirements and can save the time consumed by calculating the similarity between the pattern vertices and the data vertices. But, if the data is sparse and the degree of the pattern vertices are less, then the degree criteria can be satisfied by most of the vertices in the data graph and we can not save much on the similarity calculations. So from the results we could see that for the randomly chosen graphs, which are intuitively sparse, the improvement of the degree-based algorithm is not so obvious and the performance may not improve with increase in size of graphs. But if a dense pattern graph is chosen, from the results shown in table 4, we can conclude that the degree-based and neighbors' degree-based algorithms show improving performance with increasing size of the data graph.

Table 5. Comparing run-times for IG, DIG and NDIG for 4 different sizes of randomly generated pattern graphs and different number of fold connections.

dvn= 284 * k (all units in secs)	k = 4 dvn = 1136	k = 5 dvn = 1420	k = 6 dvn = 1704	k = 7 dvn = 1988
No connection				
IG: pvn=5	0.016	0.0294	0.0365	0.0425
DIG: pvn=5	0.012	0.0282	0.0333	0.0418
NDIG: pvn=5	0.008	0.028	0.031	0.0392
IG: pvn=8	0.0392	0.0583	0.0785	0.1014
DIG: pvn=8	0.0381	0.0574	0.0756	0.0985

NDIG: pvn=8	0.0345	0.0534	0.0714	0.0965
IG: pvn=10	0.0605	0.0845	0.1145	0.1536
DIG: pvn=10	0.059	0.081	0.1098	0.1483
NDIG: pvn=10	0.058	0.0823	0.1091	0.1471
IG: pvn=12	0.0787	0.1178	0.1638	0.2129
DIG: pvn=12	0.0782	0.1143	0.1554	0.2122
NDIG: pvn=12	0.078	0.1122	0.1565	0.2111
Connections between each pair of folds= 30				
IG: pvn=5	0.014	0.0284	0.0361	0.0434
DIG: pvn=5	0.0112	0.0234	0.0346	0.041
NDIG: pvn=5	0.012	0.0215	0.0316	0.0382
IG: pvn=8	0.0392	0.058	0.0783	0.1024
DIG: pvn=8	0.0387	0.0558	0.075	0.0982
NDIG: pvn=8	0.0355	0.0532	0.733	0.0961
IG: pvn=10	0.0592	0.084	0.115	0.1546
DIG: pvn=10	0.0578	0.0826	0.1146	0.1532
NDIG: pvn=10	0.0547	0.0818	0.1095	0.1491
IG: pvn=12	0.0782	0.1156	0.1653	0.2179
DIG: pvn=12	0.078	0.113	0.1618	0.2146
NDIG: pvn=12	0.0778	0.1122	0.1583	0.2121
Connections between each pair of folds= 60				
IG: pvn=5	0.0152	0.0286	0.0386	0.0443
DIG: pvn=5	0.014	0.0238	0.0366	0.0425
NDIG: pvn=5	0.012	0.0216	0.0342	0.0391
IG: pvn=8	0.0395	0.0604	0.0776	0.105
DIG: pvn=8	0.0378	0.0564	0.0755	0.099
NDIG: pvn=8	0.0348	0.0534	0.731	0.0977
IG: pvn=10	0.0582	0.084	0.116	0.156
DIG: pvn=10	0.055	0.081	0.117	0.153
NDIG: pvn=10	0.0535	0.077	0.109	0.149
IG: pvn=12	0.0821	0.1152	0.1645	0.2236
DIG: pvn=12	0.078	0.1167	0.1638	0.2185
NDIG: pvn=12	0.0778	0.1136	0.1635	0.2147

(table continued from pervious page)

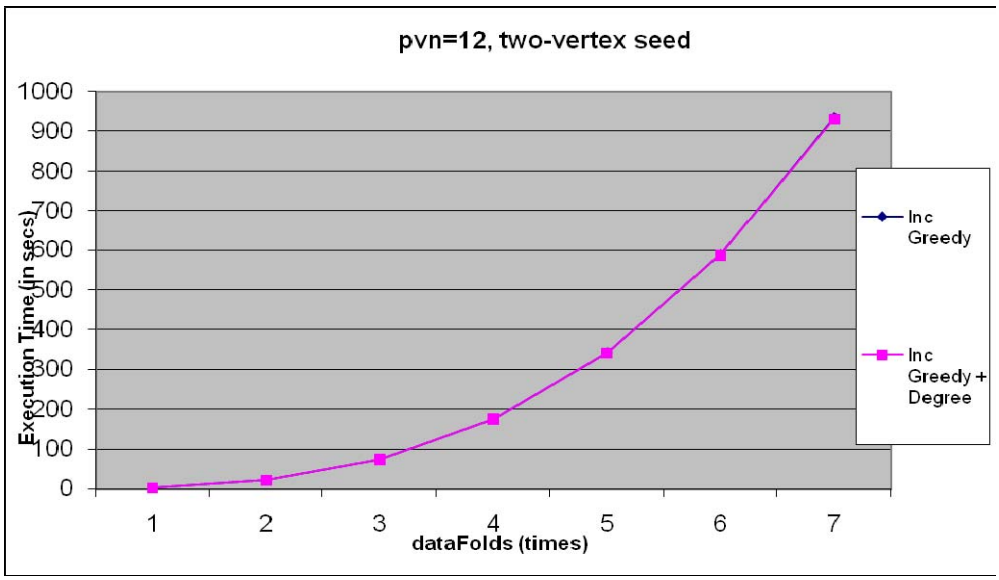

Figure 7. Comparing performances of original and Degree-based Greedy algorithms (with two-vertex seed)

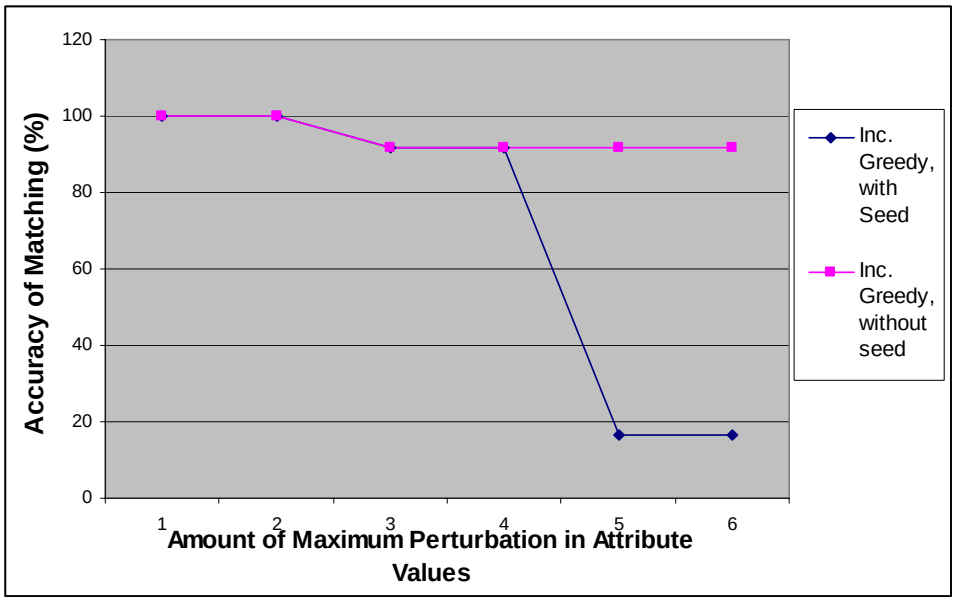

Figure 8. Comparing accuracy of Incremental Greedy algorithm with and without seed (Here accuracy is% of retrieved pattern vertices embedded in data graph)

The most important fact for the run-time performances of these algorithms with two vertex seeds is that the process of seed generation dominates the run-time. But still if we look closely, it can be noticed that the degree-based methods work faster by very small fractions of a second. Since comparison of the run-time performances of bigger seeds (say three vertices, etc.) for these algorithms will be similar to that of a two-vertex seed, we do not discuss it.

We can see from Figure 8 that with perturbation levels higher than 4 or more, the accuracy of the incremental greedy algorithm diminishes. In these cases the same incremental greedy algorithm which starts with finding an exhaustive two-vertex seed gives an accuracy of around 90 percent. It can also be mathematically proved that the computational time in Figure 4, 5 and 6 should be linearly increasing with the increase of data graph size, whereas the seed calculation in Figure 7 takes non-linear time.

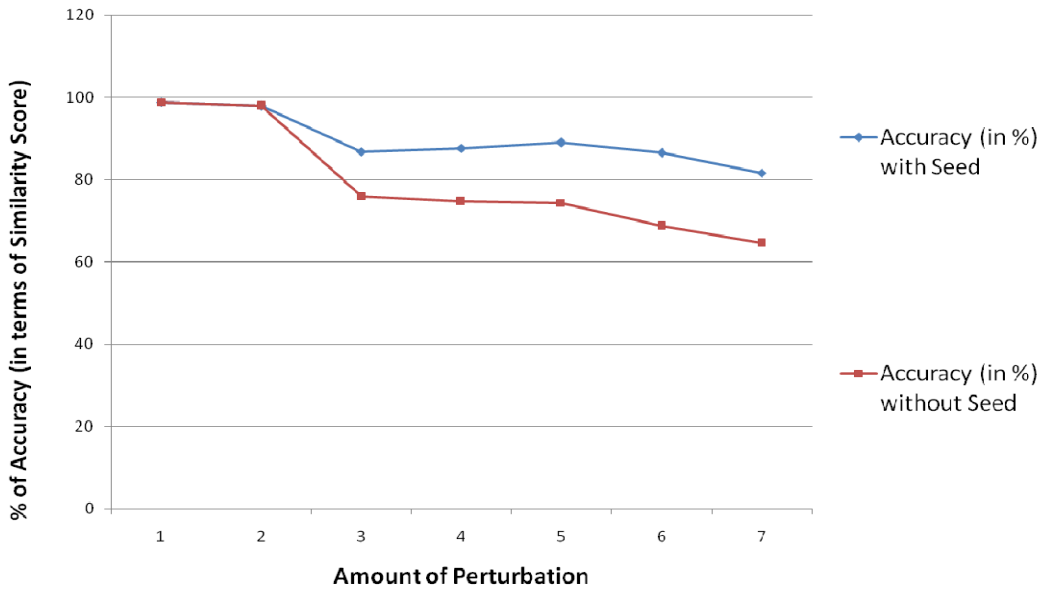

Figure 9. Comparing accuracy of Incremental Greedy algorithm with and without seed (Here accuracy is similarity scores between the retrieved graph and the originally embedded sub-graph)

4.3 Expansion of a Partial Sub-graph: Results

The experimental setup for this part enables the user to choose values for two variables. Te first one denotes the number of the nodes the partial sub-graph that the user wishes to avail

for the matching and the second one denotes the number of nodes the user wants to find through the post-matching expansion process.

After the partial matching is complete, the greedy algorithms return the matches, which are nodes of the data graph. Now the algorithm ranks the attributes of vertices and edges separately based on their average attribute values and takes top few attributes based on user discretion. Those attributes are marked as the most important attributes for vertices and edges and we search for the nodes that hold bigger values for those attributes. Any numbers of attributes from one to maximum number of attributes can be used in the next part of the program.

Then we proceed with the expansion part. Number of expanded nodes, when summed with the number of the nodes in the partial graph can exceed the maximum number of nodes in a pattern graph. It is a case where we are expanding our sub-graph beyond the apriori knowledge availed through the pattern graph. We can only verify those nodes which are created by the expansion in first V_P instances.

Since we already know our targets in the data graph once we find the potential individuals through expansion, we could try to see whether the newly found individual is supposed to be a copy of the pattern vertex embedded in the data graph. In this way for all vertices found through expansion, we measure the total goodness of the expansion, which is defined as,

Total Goodness of Expansion = Number of embedded copies of pattern vertices that found from data vertices/ Number of all the data vertices embedded in the data graph as copies of data graph (ratio expressed in percentage)

Experiments were performed based on the setup as described above and the Goodness of Expansion value (in percentage) turned out to be as follows,

Table 6. Accuracy of Expansion for different Partial Graph sizes and different Expansion sizes.

Partial Graph size	Expanded by nodes								
	2	3	4	5	6	7	8	9	10
2	100	100	100	85.71	75	66.66	60	54.54	50
3	100	100	100	100	100	89.99	81.81	75	NA

4	83.33	85.71	87.5	88.88	89.99	81.81	75	NA	NA
5	85.71	87.5	88.88	89.99	90.9	83.33	NA	NA	NA
6	87.5	88.88	89.99	90.9	91.66	NA	NA	NA	NA
7	88.88	89.99	90.9	83.33	NA	NA	NA	NA	NA
8	89.99	90.9	83.33	NA	NA	NA	NA	NA	NA
9	90.9	91.66	NA	NA	NA	NA	NA	NA	NA
10	91.66	NA	NA	NA	NA	NA	NA	NA	NA

All values in the table are % of Accuracy of the Expansion
NB: We have not exceeded the expansion by the original pattern size of 12

(table continued from previous page)

The results show that it can find most of the copies of the pattern vertices from the data vertices. It seldom finds data vertices which are not copies of the pattern vertices. But since the Goodness of Expansion remains high (which means that those vertices also have high vertex and edge attribute values) and as also the same vertices keep coming back as a part of different expansions, we could conclude that those unidentified nodes are also potential threat elements which are not included in the pattern graph. Hence it could be said that the post-matching expansion algorithm works with a high success rate.

5. Conclusion and Future Work

In this thesis, two incremental greedy algorithms for approximate sub-graph matching for attributed graphs have been presented and compared, which can be used as a component of social network analysis for terrorists' group detection. This thesis proposes to use vertex degree information to reduce the computational cost during the greedy search. Experimental results using graphs constructed from a simulated terrorist activity database suggest that the degree-based incremental greedy algorithm is much more efficient while maintaining similar level of graph matching accuracy. If the data is noisy, instead of using the incremental greedy algorithm directly, it may be useful to sacrifice run-time to predict more accurate results by using appropriate seeds.

- One aspect of future work could be the post-matching expansion of the partial pattern. This problem can be approached in different ways to see if any further improvements could be made. We have only tried it for the original set of data graph, which can be increased by many folds (as we did while testing the greedy algorithms) and also noise can be introduced to see how the post-matching expansion reacts to the noise and to what degree.

- Due to time limitation we could not explore the opportunities to setup role mining heuristics to find the role of an individual in the data graph. This information could help making smart decisions to facilitate network disruption or other social network related processes.

- Network disruption is probably one of the most interesting and promising study that could arise from this thesis. Once a match for a pattern is found we could build heuristics that could successfully detect cut-vertices and hence disconnect the graph. Depth-first or Breadth-first search could be used to detect the graph's connectivity before and after the cut-vertices have been remove. Disconnection efficiency can also be a prospective topic of study.

- In our thesis, although we have selected some threat objects (like, a terrorist or a criminal) to create the pattern graph, we have not used information of the entire set of threat objects. A study of these threat objects can be useful to find efficient methods to profile terrorists.

- We could not explore the efficiency of the greedy algorithms for any data graph of more than approximately 2000 nodes. The limitation on the memory block usage in C compiler restricts us to do the testing. But these experiments can be performed on clusters or supercomputers in a distributed environment. The program might need to go through modifications to be executed under a distributed environment and also it remains to be seen if, under those circumstances, the time taken for the communication between processors overwhelms the original efficiency of the algorithms.

- One could also explore the possibility of weighting certain attributes and also playing with the weights of the vertex similarity and edge similarity in the cumulative objective function.

- In this thesis we do not mention how we could apply the discussed methods for a dynamic graph.

Due to time limitation we could not perform all the experiments we would have liked to perform. Also because of the uniqueness of the graph similar tests in another dataset(s) could substantiate the results.

Bibliography

[1] "The Structural Analysis of Criminal Networks" by D. McAndrew. The Social Psychology of Crime: Groups, Teams and Networks. 1999, D. Canter and L. Alison, Eds Dartmouth Publishing, Aldershot, UK, 53-94.

[2] "The Application of Network Analysis to Criminal Intelligence: An Assessment of the Prospects" by M. K. Sparrow. Social Networks 1991, Vol. 13, Pages 251-274.

[3] "Mapping Networks of Terrorist Cells" by V.E. Krebs. Connections 24(3), 43-52 (2001).

[4] "The Key Player Problem" by S. Borgatti. Proceedings from National Academy of Sciences Workshop on Terrorism, Washington DC (2002)

[5] "Sociological Skills Used in the Capture of Saddam Hussein" by V. Hougham. http://www2.asanet.org/footnotes/julyaugust05/fn3.html (2005)

[6] "Organized Crime: A Social Network Approach" by J.S. McIllwain. Crime, Law and Social change 32, 301-323 (1999).

[7] "Social Network Analysis: Methods and Applications" by Wasserman, S., Faust, K. Cambridge University Press, Cambridge (1994)

[8] "Analyzing Terrorist Networks: A Case Study of the Global Salafi Jihad Network" by J. Qin, Jennifer J. Xu, Daning Hu, Marc Sageman and Hsinchun Chen. Kantor, P., Muresan, G., Roberts, F., Zeng, D.D., Wang, F.-Y., Chen, H., Merkle, R.C. (eds.) ISI 2005. LNCS, vol. 3495, pp. 287–304. Springer, Heidelberg (2005).

[9] "Centrality in Social Networks: Conceptual Clarification" by L.C. Freeman. Social Networks 1, 215–239 (1979)

[10] "The Social Organization of Conspiracy: Illegal networks in the heavy electrical equipment industry" W.E. Baker, R.R. Faulkner. American Sociological Review 58(12), 837–860 (1993).

[11] "The Network Paradigm Applied to Criminal Organizations: Theoretical Nitpicking or a Relevant Doctrine for Investigators?" by P. Klerks. Connections 24, 53-65.

[12] "The Application of Link Analysis to Police Intelligence" by W. R. Harper and D. H. Harris. Hum. Fact. 17, 157-164.

[13] "The Analysis of Social Networks" by R.L. Breiger. In Handbook of Data Analysis 2004, M. A. Hardy, A. Bryman, Eds. Sage Publications, London, UK 505-526.

[14] “Authoritative Sources in a Hyperlinked Environment” by J. M. Kleinberg 1999. J. Assoc. Comput. Mach. 46, 604-632.

[15] “Mining the Network Value of Customers” by P. Domingos, M. Richardson. 2001, Proceedings of the seventh ACM SIGKDD, P 57-66.

[16] “Applying Social Network Analysis to the Information in CVS Repositories” by L. Lopez-Fernandez, G. Robles, J.M. Gonzalez-Barahona. 2004, International Workshop on Mining Software Repositories, P 101-105.

[17] “Data Mining in Social Networks” by David Jensen, Jennifer Neville. 2003, Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers, P 289-303.

[18] “Graph-based data mining for social network analysis” by M. Mukherjee and L. B. Holder. In Proceedings of the ACM KDD Workshop on Link Analysis and Group Detection, 2004.

[19] “Community Mining from Multi-relational Networks” by D. Cai, Z. Shao, X. He, X. Yan, J. Han. Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, 2005, vol. 3721, pp. 445-452.

[20] **“**Detecting Key Players in 11-M Terrorist Network: A Case Study” by Nasrullah Memon and David L. Hicks. Third International Conference on Availability, Reliability and Security, 2008. P 1254-1259.

[21] **“**Inexact graph matching by means of estimation of distribution algorithms” by Endika Bengoetxea, Pedro Larrañaga, Isabelle Bloch, Aymeric Perchant and Claudia Boeres. Pattern Recognition, Volume 35, Issue 12, December 2002, Pages 2867-2880.

[22] **“**Parallel Algorithms for Labelled Graph Matching” by A. M. Abdulkader. PhD dissertation, C Colorado School of Mines, 1998.

[23] “A Randomized Heuristics for Scene Recognition by Graph Matching” by Maria C. Boeres, Celso C. Ribeiro and Isabelle Bloch. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 3059/2004, Pages 100-113.

[24] “Degree-based Approximate Sub-graph Matching for Social Network Analysis in Terrorist Detection” by Partha Basuchowdhuri, Dr. Jianhua Chen, Dr. Peter P. Chen (Submitted to ICAI 2009).

Vita

Partha Basuchowdhuri was born and raised in Calcutta (now Kolkata), India. He attended Jadavpur Vidyapith, Ramakrishna Mission Vidyalaya (Narendrapur) and Jodhpur Park Boys' School to complete his primary, secondary and higher secondary studies respectively. He graduated from Bengal Engineering College- Shibpur (now B.E.S.U) in 2003, with a first-class Bachelor of Engineering degree in Electronics and Communication Engineering. He completed his master's degree from the Electrical and Computer Engineering Department of Louisiana State University in May, 2008.

He is presently in the process to complete his master's degree from the Department of Computer Science at Louisiana State University by May, 2009. Mr. Basuchowdhuri is interested to continue his research as a Ph.D. researcher in the area of data mining. His primary area of interests are clustering, graph-mining, pattern recognition, bayesian learning and other related topics in data mining and machine learning.

His main extra-curricular activities involve playing soccer, painting, photography and learning culinary skills from people of various cultural backgrounds.