

RANKS AND PARTIAL CUTS IN FORWARD HYPERGRAPHS

by

REGINALD ELIAS SAWILLA

A thesis submitted to the School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada

April 2011

Copyright © Reginald Elias Sawilla, 2011

Abstract

Many real-world relations are networks that can be modelled with a kind of directed hypergraph named a forward hypergraph (F-graph). F-graphs capture the semantics of both conjunctive and disjunctive dependency relations. Logic statements are sometimes represented using AND/OR directed graphs and they directly correspond with F-graphs; we provide algorithms to convert between the two types of graph.

One problem of interest in networks is determining the degree to which the network, with a priority on certain elements, depends upon individual nodes. We address this problem by providing an algorithm, AssetRank, which computes vertex ranks and takes into account network priorities, preferential dependencies, and extra-network influences.

A second problem of interest in networks is optimizing the removal of nodes to separate two subcommunities (source and target) to the greatest practical degree, even when a complete disconnection is impractical. The problem is complicated by the need to consider the cost of removing nodes, a budget that constrains the degree to which separation is possible, cascading effects of removing a node, non-linear effects of removing nodes in combination, and removing nodes with the

greatest impact on the subcommunities. To this end, we use F-graphs and introduce the concepts of vertex closures and closure-relation graphs. We created two partial-cut algorithms: the first one computes an optimal solution to this NP-hard optimization problem, and the second one estimates an optimization by exploring the closure-relation graph in a best-first search manner.

Computer network defence provides a ready application area. Network defenders wish to understand which services and hosts are defence priorities (defence goals), and then, which configurations and vulnerabilities are the most useful to attackers in reaching the defence goals. The defenders' resources are constrained in terms of available person-hours, finances, and acceptable impacts to operations. The work in this thesis supports network defenders by providing actionable information that efficiently removes attack enablers and ensures defence priorities. We present an integration of our algorithms with commercial and open-source network security software.

Co-authorship

A presentation of the material in Chapter 3 and the related background material was published at ESORICS 2008 [73] with Dr. Ou. The author of this thesis is the primary author of that publication and a related Technical Memorandum [72]. Sawilla's contributions were the idea of applying AssetRank to exploit-dependency attack graphs, extension of the AssetRank algorithm to compute rank differently for AND vertices and OR vertices, assignment of parameter weights, extension of the AssetRank algorithm to use a damping matrix, proof of convergence, and an interpretation of the damping factor for attack graphs. Ou adapted his MulVAL attack graph generator to calculate arc weights for input to AssetRank, and designed the experiments. The text was written collaboratively in a series of refinements. Both authors developed the interpretation for vertex weight and arc weight.

An early version of material in Chapter 4 and the related background material was published at the North Atlantic Treaty Organization (NATO) Symposium on Information Assurance and Cyber Defence [71] with Dr. Burrell. The author of this thesis is the primary author of that publication and a related Technical Memorandum [70]. Sawilla's contributions were the idea of a closure of a course-of-action (called the closure of a vertex set in this work) along with the idea and implemen-

tation of the algorithms. The text was written collaboratively in a series of refinements.

Acknowledgments

I thank my supervisor, Dr. Skillicorn, for his impressive insights. Our meetings were always tremendously valuable and each one led me to chastise myself for not meeting earlier. My only complaint is that he reviewed my drafts at a supernatural speed and so any breathers were short-lived.

I sincerely appreciate the careful reading of my thesis by the other examiners (Drs. Knight, Dawes, Dean, and Rappaport). Their diverse perspectives and detailed comments enhanced the thesis in many ways.

Working with my collaborators, Drs. Burrell and Ou, was a genuine pleasure. Their character, enthusiasm, intelligence, and friendship, made even the eleventh-hour paper revisions entirely enjoyable.

I am deeply grateful to my manager, Dr. Lefebvre, and my employer, Defence Research and Development Canada, for generously providing me the opportunity to fulfill this lifetime goal. The unwavering support of Dr. Lefebvre and my colleagues was a great encouragement on many occasions.

I especially appreciate my wife, Darcie, who has been a tremendous help and strength during my entire university education. She staunchly bore a considerable load to provide a high quality of life for our family. Likewise, I am indebted to my

children, Shafir, Aliya, Ciara, and Eve, for their interest, joyful support and love. My entire extended family has been a tremendous blessing and I thank them for their enthusiasm and continual encouragement.

Above all, I esteem Christ's example of the potential of humankind and the accompanying humility that such a mediation entails.

Dedication

I dedicate this thesis to:

My parents: Who I am forever indebted to for their example of hard work, perseverance, encouragement, and faith.

My wife: Who is the most important person in my life and my best friend.

My children: Who are boundless sources of joy and pride.

Statement of originality

I hereby certify that the research presented in this dissertation is the original work of the author. Ideas and techniques from the work of others are fully acknowledged in accordance with standard referencing practices.

Table of contents

Abstract	i
Co-authorship	iii
Acknowledgments	v
Dedication	vii
Statement of originality	viii
Table of contents	ix
List of tables	xii
List of figures	xiii
List of algorithms	xv
Chapter 1 Introduction	1
1.1 Assets	4
1.1.1 Dynamism	5
1.1.2 Asset relations	5
1.2 Problems	6
1.2.1 Ranking AND/OR directed graphs	6
1.2.2 Partial cuts in AND/OR directed graphs	6
1.3 Example application — computer network defence	8
1.4 Contributions	12
1.5 Organization of thesis	15
Chapter 2 Graph theory concepts	16
2.1 Types and attributes	16

2.1.1	Undirected graphs	17
2.1.2	Vertex and edge weights	17
2.1.3	Directed graphs	19
2.1.4	Hypergraphs	20
2.1.5	Directed hypergraphs	23
2.1.6	AND and OR vertices	25
2.2	Matrix representations	27
2.2.1	Affinity matrix	29
2.2.2	Laplacian matrix	32
2.3	Spectral graph theory	37
2.3.1	Edge cuts	38
2.3.2	Vertex ranking	41
2.3.3	Link prediction	43
2.4	Power-law degree distributions	44
2.5	Decision support for computer network defence	45
2.5.1	Attack graph semantics	45
2.5.2	Attack graph ranking	46
2.5.3	Graph cuts and course of action decision support	47
2.6	Supermodular knapsack problem	51
Chapter 3 Vertex ranking		54
3.1	Introduction	54
3.2	Conversion between F-graphs and AND/OR digraphs	60
3.3	AssetRank for AND/OR digraphs	60
3.3.1	AND vertices	63
3.3.2	Convergence	64
3.3.3	Vertex-specific damping	66
3.3.4	Personalization vector	67
3.4	Parameter assignment	68
3.4.1	Dependency matrix (D)	68
3.4.2	Damping matrix (Δ)	71
3.4.3	Personalization vector (P)	73
3.5	Stochastic interpretation of AssetRank	73
3.6	Summary	76
Chapter 4 Partial-cut vertex sets		78
4.1	Introduction	78
4.2	Generation of scale-free networks	82
4.3	Closure of AND/OR directed graph vertex set	88
4.3.1	Discussion and definition	88

4.3.2	Vertex set closure algorithm	90
4.4	Partial-cut vertex sets in AND/OR directed graphs	91
4.4.1	Discussion and definition	91
4.4.2	Alternative rank and loss costs	93
4.4.3	Brute-force partial graph cut algorithm	93
4.4.4	Closure-relation graphs	95
4.4.5	Best-first search partial-cut algorithm	99
4.4.6	Equivalent removals	102
4.4.7	Redundant deletions	104
4.5	Complexity	110
4.5.1	Vertex set closure	110
4.5.2	Brute-force partial-cut vertex set	110
4.5.3	Best-first search partial-cut vertex set	111
4.5.4	Provable goodness bounds	112
4.6	Summary	114
Chapter 5 Experiments		115
5.1	AssetRank experiments	115
5.1.1	Algorithm evaluation plan	116
5.1.2	Experiment 1: Arc weights and damping factors	117
5.1.3	Experiment 2: 5-host network	120
5.1.4	Experiment 3: Control-system network	125
5.1.5	Discussion	126
5.2	Partial-cut vertex set experiments	129
5.2.1	Algorithm evaluation plan	129
5.2.2	Experiment 1: Simple 3-host network	130
5.2.3	Experiment 2: Scalability and accuracy	133
5.2.4	Experiment 3: Full cycle integration	136
5.2.5	Network architecture	137
5.2.6	Implementation decisions	139
5.2.7	Interface and attack scenario	140
5.2.8	Trimmed scenario	148
Chapter 6 Conclusion		150
6.1	Summary	150
6.2	Limitations and suggested work	155
Bibliography		159

List of tables

Table 1.1	Vertex costs for Figure 1.3 graph	12
Table 2.1	Graph matrices	28
Table 2.2	Hypergraph and undirected graph clustering are equivalent	34
Table 3.1	CVSS exploitability metrics and success likelihoods	69
Table 4.1	Distribution of incoming subnet connections	86
Table 4.2	Distribution of outgoing subnet connections	86
Table 5.1	AssetRanks for Experiment 1a	119
Table 5.2	AssetRanks for Experiment 1b	119
Table 5.3	AssetRanks for Experiment 2a	121
Table 5.4	AssetRanks for Experiment 2b	122
Table 5.5	Vertex costs for Figure 5.9 graph	133
Table 5.6	Comparison of Brute-Force and Best-First Search	136
Table 5.7	Ratio statistics for partial cuts with non-optimal rank-sums	136
Table 5.8	Algorithm time and accuracy comparison	149

List of figures

Figure 1.1	Computer network defence workflow	9
Figure 1.2	Example network	10
Figure 1.3	Attack graph for Figure 1.2 example network	11
Figure 2.1	Example of an undirected graph — social network	17
Figure 2.2	Example of a directed graph — web community	19
Figure 2.3	Example of a hypergraph — travellers	21
Figure 2.4	Clique expansion undirected graph of Figure 2.3	21
Figure 2.5	Undirected graph equivalent to Figure 2.3	22
Figure 2.6	Example of a directed hypergraph — attack graph	24
Figure 2.7	Example of AND/OR graph — attack graph	27
Figure 3.1	An example network from Ingols <i>et al.</i> [39]	55
Figure 3.2	Attack graph for the network in Figure 3.1	56
Figure 3.3	AssetRank computation for an AND/OR graph	63
Figure 4.1	Random 30-host scale-free computer network	87
Figure 4.2	Effects of vertex removal	89
Figure 4.3	Closure-relation graph for attack graph in Figure 4.2a	96
Figure 4.4	Alternate representation of graph in Figure 4.3	97
Figure 4.5	Random 10-host scale-free computer network	106
Figure 4.6	Attack graph for the network in Figure 4.5	107
Figure 4.7	Closure-relation graph for attack graph in Figure 4.6	108
Figure 4.8	Magnified section of Figure 4.7	109
Figure 5.1	Scenario for experiments 1a and 1b	117
Figure 5.2	Ranked attack graph for the Experiment 1a scenario	118
Figure 5.3	Ranked attack graph for the Experiment 1b scenario	120
Figure 5.4	Ranked attack graph for the Experiment 2a scenario	123
Figure 5.5	Ranked attack graph for the Experiment 2b scenario	124
Figure 5.6	A realistic network scenario for Experiment 3	126

Figure 5.7	Ranked attack graph for the Experiment 3 scenario	128
Figure 5.8	Example 1 network	130
Figure 5.9	Ranked attack graph for Example 1 network	131
Figure 5.10	Genesis network	138
Figure 5.11	Genesis: Observe phase screenshot	143
Figure 5.12	Genesis: Orient phase screenshot	144
Figure 5.13	Genesis: Decide (AssetRank) phase screenshot	145
Figure 5.14	Genesis: Decide (COA) phase screenshot	146
Figure 5.15	Genesis: Act phase screenshot	147

List of algorithms

3.1	Convert F-graph to AND/OR directed graph	61
3.2	Convert AND/OR directed graph to F-graph	61
4.1	VertexSetClosure(G,C,S,T) — Closure of a vertex set	91
4.2	BruteForcePCVS(G,S,T,Budget) — Partial-cut vertex set	95
4.3	BFS-PCVS(AG,S,T,Budget) — Best-first search PCVS	101

Chapter 1

Introduction

This thesis describes Forward Hypergraphs (F-graphs), introduces two problems of interest in F-graphs, and shows applications that the graph problems have to problems of interest in networks. We describe the state of the art in the problem areas, and propose a proof-of-concept for computer networks that demonstrates the usefulness and practicality of our approach.

Many real world relations can be viewed as a network. Examples include:

- Social network: relationships between people
- Social influence network: influence potential between people
- Biological contagion network: contagion spread between people
- Computer network: logical and physical connections between computers
- Malware contagion network: computer worm and virus propagation between computers

- Service network: interdependencies between computer services

A graph is a mathematical abstraction of the details of possible interactions and dependencies in many real-world networks. The graph vertices represent objects (tangible or intangible) and the edges represent relationships between the objects. The edges can be directed, which enables the specification of both asymmetric and symmetric relationships. In a computer service network, the vertices represent services like email and file storage while the edges represent when one service directly depends upon another service in order to function. F-graphs, in particular, are well-suited to capture the details of dependencies in networks.

Edges in directed hypergraphs (see, for example, Gallo *et al.* [28]) can have any number of sources and targets, in contrast with directed graphs (digraphs) where edges have a single source and target. An F-graph is a directed hypergraph in which each edge is restricted to have one source. An edge with multiple targets represents a conjunctive dependency that the source vertex has on the target vertices. A formal definition of F-graphs and their relationship with AND/OR directed graphs is given in Section 2.1.5.

A graph representation enables the application of powerful analytical techniques to give insight into network problems. An analyst can use a visual representation of small graphs to better understand connections and dependencies. Vertices that are essential to the graph are identifiable. In a biological contagion network, a single vertex that connects two graph regions corresponds to a choke point for the contagion spread between two communities. If a person who is identified as vital to the contagion spread is inoculated, the contagion could be prevented from spreading to a neighbouring community.

However, even for moderately sized networks, graph visualizations are often too large and complex for a human to fully comprehend. While an analyst will quickly understand that connections exist between multiple regions of the graph it is very challenging to know which graph vertices are significant. This is due to the many interconnections, the different strengths of connections, multiple paths between vertices, and cycles (loops) in graphs. For instance, an attack graph is a graph representing the ways an attacker can move in a network from an initial computer, through intermediary computers, to target locations. An analyst would like to select critical computers and services to defend (defence goals) and then determine the privileges and vulnerabilities that are the most important to an attacker's success in reaching the defence goals. Even for a small computer network the attack graph may contain thousands of vertices and so it is impractical for an analyst to determine the most important privileges and vulnerabilities from a simple graph visualization.

The aim of this thesis is to address two particular graph problems and apply the work to computer network defence. The first problem addressed is obtaining a rank metric for F-graph vertices that reflects the degree to which they are depended upon by the network, which has a user-selected priority on a set of vertices. The second problem addressed is maximally disrupting connectivity between two communities in an F-graph by spending a budget to remove vertices in an optimized manner.

In a health network, the first problem corresponds with identifying locations or people that most enable contagion spread between subcommunities, or where contagions are likely to cumulate due to contact with others in the network. The second problem corresponds with efficiently spending limited resources by quar-

antining key locations or inoculating specific people in order to maximally disrupt the ability of a contagion to spread from a source group to a target group.

For computer network defence, the first problem corresponds with identifying vulnerabilities, services, and connections that an attacker is most likely to use in order to reach specific targets. The second problem corresponds with efficiently spending resources and minimizing user impact by choosing patches to apply, services to shut down, or network connections to remove in order to maximally disrupt attackers' abilities to reach critical hosts.

Security analysts in the fields of computer network defence, public safety, and counter-terrorism require tools that can distill the overwhelming amount of information into an understanding of which vertices are strategically important, and how sets of vertices form critical combinations. This knowledge would help them to efficiently employ limited human and financial resources.

1.1 Assets

The first problem we address in the thesis is to determine the ranking of assets according to their importance to the functioning of a system. The ranking is always done within a context, whether it be the security contexts of confidentiality, integrity and availability, or, the political contexts of influence and public opinion. The rank metric gives a direct indication of the comparative importance of assets in the network.

We define an asset to be any object that is of interest to the system. An object is a person, geographical location, tangible item, or concept, that has value to the system. The collection of all assets form a system.

1.1.1 Dynamism

One challenge in ranking assets is that their importance changes as the operational needs and priorities of the network change. For example, suppose that a nation state is preparing to launch air strikes against certain targets. In the weeks leading up to the mission, the confidentiality of the site selection is extremely important. Once the mission is underway, the confidentiality of the site selection is not at all important because the air strikes reveal the sites selected but now the availability of the targeting systems is critical. A system for dynamic asset ranking and course-of-action (COA) decision support must be able to rapidly revalue assets, taking into account the current priorities of the organization.

1.1.2 Asset relations

Asset relations are naturally suited to a graph representation. If an asset a_1 depends upon asset a_2 in some context then a_1 and a_2 can be considered as vertices and the dependency is a directed edge from a_1 to a_2 . The relations in each context (for example, confidentiality, integrity, or availability) can be very different, and so each context is represented by a different graph. A graph representation that can take into account compound dependencies and account for redundancies is an AND/OR directed graph where AND vertices encode compound dependencies and OR vertices encode alternative choices [3]. F-graphs are equivalent to AND/OR directed graphs and we use the two terms interchangeably. In Sections 2.1.5 and 2.1.6 we explain these graphs in greater detail.

1.2 Problems

1.2.1 Ranking AND/OR directed graphs

We build upon related work that ranks (OR only) directed graphs. PageRank [67] and Hyperlink-Induced Topic Search (HITS) [46] are seminal algorithms that introduced vertex ranking for directed graphs in the context of web pages. Those algorithms, along with the Stochastic Approach for Link-Structure Analysis (SALSA) [50], AuthRank, HubRank, and SymRank [20], are applications of spectral graph theory. Besides being used for ranking web pages, the algorithms have been applied to clustering autonomous systems on the Internet, and predicting areas of link stress.

AND/OR directed graphs have divergent semantics from directed graphs and require different algorithms. In this thesis we introduce AssetRank, a ranking algorithm designed for these unique semantics.

1.2.2 Partial cuts in AND/OR directed graphs

The second problem we address in the thesis is finding a method to maximally disrupt connectivity between two graph subcommunities. We extend the following established concepts in graph theory. A k -connected graph is a graph where k is the least number of vertices whose deletion will disconnect the graph. Local vertex connectivity of vertices u and v gives the least number of vertices that will disconnect u and v . We extend these terms to consider the optimal set of vertices, whose cost does not exceed k , that will maximally (though not necessarily completely) disrupt the connectivity between a source and target set in a ranked graph. We first introduce our definition of the term *induced*.

Definition 1.1 (Induced). Let $G = (V, A)$ be an AND/OR directed graph (F-graph) where each vertex has a cost and rank weight. Let S (source) and T (target) be disjoint subsets of V (representing two distinct subcommunities). The graph *induced* by S and T on G is the minimal graph that contains S , T , edges on all paths from S to T , and all vertices on those paths.

Let $G = (V, A)$, S , and T be as defined in the above definition. The vertex cost weight reflects the cost of removing that vertex from the graph. For example, in a biological contagion network, V is the set of people who have not been inoculated, S is the set of contagious people, and T is a set of people whose health must be ensured. The vertex cost is the cost of inoculating a person, thus removing them from the contagion network. The goal is to maximally prevent the contagion spread to T under the constraint that inoculation resources are limited. Let G' be the subgraph of G induced by S and T . We want to maximally disrupt connectivity from S to T at the lowest cost. The cost of removing a set of vertices D from the graph is given by the sum of their cost weights and connectivity is maximally disrupted when the sum of vertex rank weights in the minimal graph induced by S and T on $G' - D$ is minimized. Note that the removal set D may contain vertices in S or T .

While the aim of the existing concept of local vertex connectivity is to disconnect two vertices, our goal is to maximally disrupt two sets of vertices. A total disconnection is the maximum possible disruption but, in practice, a total disconnection is often not possible so we have assumed the goal of maximally, though not necessarily completely, disrupting connectivity. We had to select a measure of disruption and for that purpose we chose the sum of the rank value of the remaining vertices as our indicator of the disruption between the graph communi-

ties since each vertex rank value directly reflects the dependence of the network on the object represented by the vertex. Removing the maximum amount of rank corresponds with maximally disrupting the dependencies between the source and target subgraphs.

1.3 Example application — computer network defence

This graph problem has an application in the field of network security where network defenders have finite resources. For instance, defenders of enterprise computer networks operate with a fixed quantity of staff resources available to test and roll-out configuration changes and patches. Defenders cannot assume that all attacks will be completely neutralized so they must determine the course of action (COA) that maximally defends the network. The defender wishes to spend his/her resources in the manner that maximally disrupts the attackers' capability to control the defender's services, according to the defender's priorities. We recognize that the attackers' priorities might not be the same as the defender's priorities; however, our prime concern is the defence goals. The attackers' priorities will likely be diverse based on their differing motivations and they need not distract the defender, whose responsibility is to the organization's priorities. The degree to which an attacker depends upon each attack asset (for example, user credentials, network connectivity, and vulnerabilities) in a network attack graph is represented by rank values computed for each attack asset. The cost to change a network configuration is provided by the defender and corresponds with removing that capability from the graph, thus denying its use by the attacker. We do not assume the attack can be completely removed within the budget. The problem is computing a

COA, in a time that is polynomial with the network size, which removes the greatest amount of attacker capability to reach prioritized defence goals, while staying within the defender’s resource budget.

Throughout this thesis, we illustrate the use of AssetRank and partial F-graph vertex cuts by applying the research to computer network defence (CND). Figure 1.1 shows the CND work flow from gathering information about the network through to implementing changes. The shaded areas, namely “Ranks” and “Compute Courses of Action” are the areas where the research in this thesis fills existing gaps. AssetRank computes ranks for the attack graph vertices which reflect the importance of each vertex to an attacker in reaching network assets the defender wants to protect. Our work on partial cuts is used to compute courses of action which increase the security of the network.

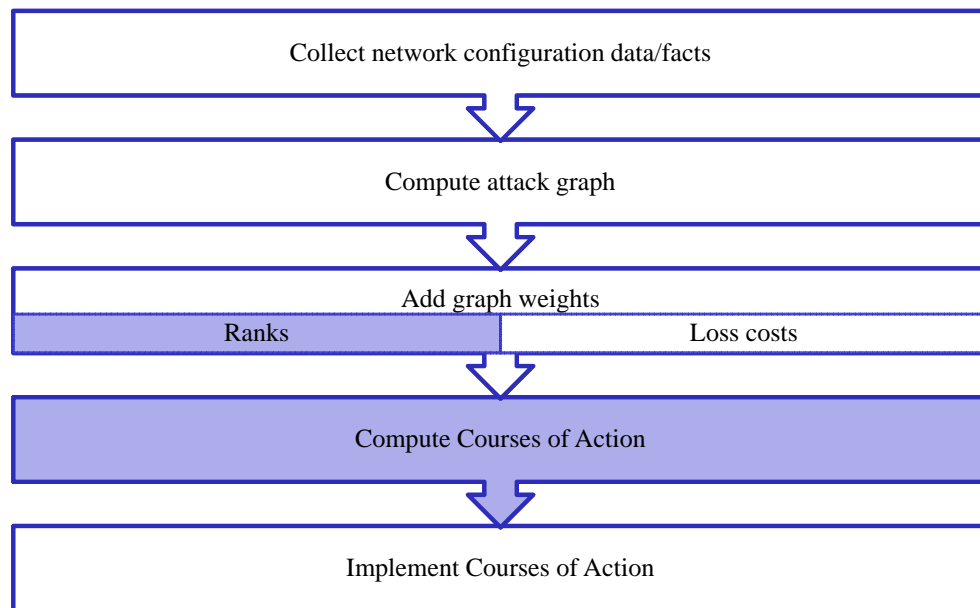


Figure 1.1: Computer network defence workflow. Shaded regions indicate applications of the theory in this thesis.

The network in Figure 1.2 has 3 hosts plus the attacker's PC. Each computer has a vulnerability which the attacker can exploit remotely. The likelihood of successfully exploiting each vulnerability differs due to the type of vulnerability and the attack tools available. The likelihoods of successful exploitation are: web server – 80%; mail server – 40%; and file server – 100%. The attacker does not have direct access to the file server but can reach it by a multistep attack through either the web server or the mail server.

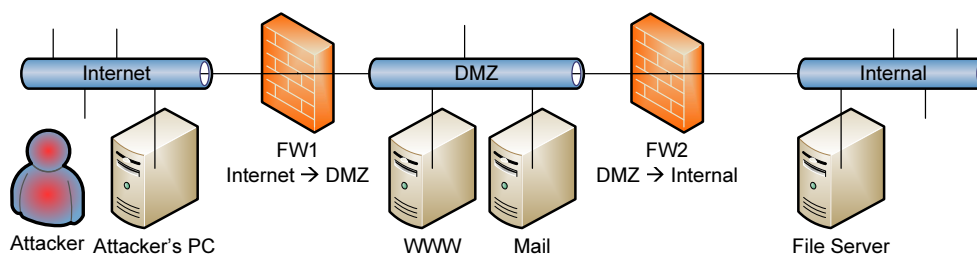


Figure 1.2: Example Network: The attacker's goal is to gain control of the file server.

Figure 1.3 is the attack graph containing the possible attack steps for this scenario. The attack graph was generated using the open source MulVAL [66] framework. Vertices represent attack assets (vulnerabilities, connectivity, and stepping stones) that the attacker can use to further his attack. The arcs indicate dependencies that each derived asset has on other assets. The graph gives a proof tree for how the asset `execCode(fs, serviceaccount)`¹ is derived from other facts.

Vertex costs are given in Table 1.1 and reflect the following assumptions. Assume that no patch exists for the vulnerability on the file server so the patch has an infinite cost (vertex 29). The file server is critical to operations so its availability is the highest priority and thus shutting down the service incurs a cost of 10,000

¹`execCode(fs, serviceaccount)` states that the attacker can execute arbitrary code on the file server (fs) at the privilege of serviceaccount.

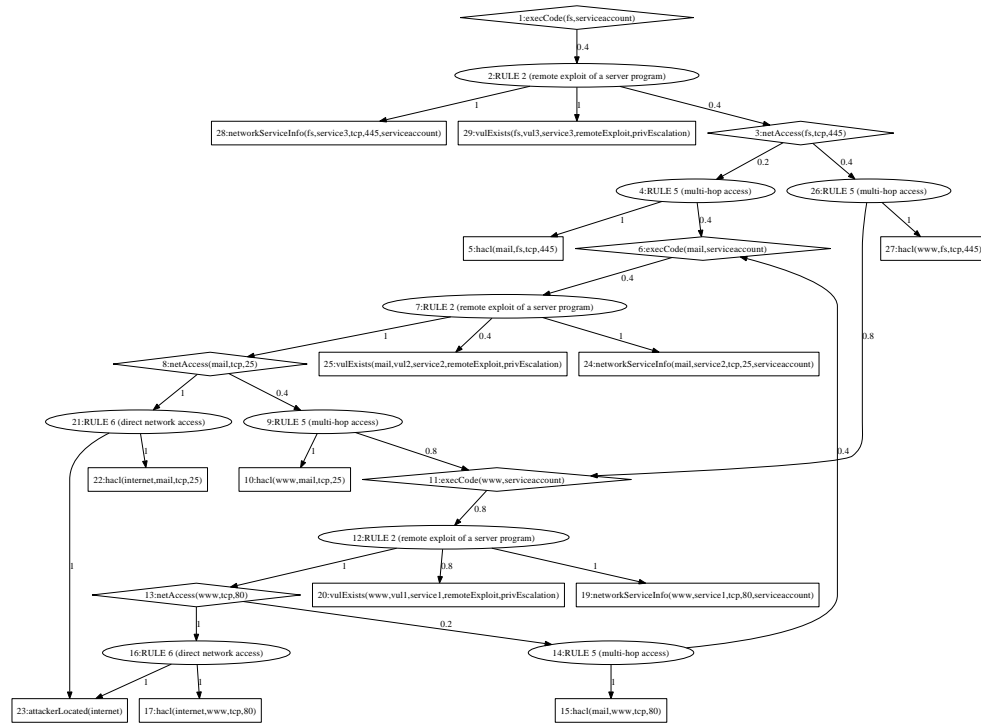


Figure 1.3: Attack graph for Figure 1.2 example network: The graph gives a proof tree for how the asset $\text{execCode}(fs, \text{serviceaccount})$ is derived from other facts. Diamonds indicate OR vertices, ellipses indicate AND vertices, and boxes indicate SINK vertices. This figure is hard to read at this scale in the printed version but can be zoomed in an electronic version.

units (vertex 28). Also assume that the vulnerability on the mail server has a well-tested patch available; thus, the cost to deploy is 100 units (vertex 25). However, the web server vulnerability only has a workaround available that introduces a risk to operations and incurs maintenance costs so the cost is 2,000 units (vertex 20). Other costs in the table are consistent with these assumptions.

While in theory the vulnerability success likelihoods, attack graph, and costs give a defender the information s/he needs to decide upon an optimal course of action, in practice the graph and other data quickly become too complex to use.

Vertex #	Description	Cost to remove (operational & physical)
5	route from mail to fs	500
10	route from www to mail	500
15	route from mail to www	10 (link is not necessary)
17	route from internet to www	3000
19	www service	3000
20	patch www server vulnerability	2000
22	route from internet to mail	750
24	mail service	1000
25	patch mail server vulnerability	100
27	route from www to fs	500
28	file sharing service	10000
29	patch file server vulnerability	∞ (no patch available)

Table 1.1: Vertex costs for Figure 1.3 graph

Tools are required to provide a ranking of the importance of each attack asset, and an analysis of the most cost-effective manner to maximally defend the network. The tools must take into account the preferences an attacker will have for the vertices in the attack graph (reflecting the ease of exploitation), graph-wide dependencies on attack assets, and remediation costs. The tools must also include the realization that operational priorities and resource constraints may be such that total defence is not possible.

1.4 Contributions

The thesis provides a ranking algorithm, AssetRank, which can handle the semantics of vertices and arcs of AND/OR directed graphs. As well, algorithms were created which minimize connectivity between graph subcommunities. Our specific

contributions in this thesis are:

Contributions related to ranking

1. Create AssetRank, a ranking algorithm that generalizes the PageRank algorithm and deals with the semantics of AND/OR directed graphs. Prove the AssetRank algorithm converges to a solution (Section 3.3).
2. Generalize PageRank's single system-wide damping factor to a per-vertex damping factor. This generalization accurately models the various likelihoods of lowered dependence on a vertex due to dependencies not captured by the graph. For example, in an attack graph, the per-vertex damping factors model an attacker's likelihood of obtaining privileges through means not captured in the graph (out-of-band attacks) (Section 3.3.3 and 3.4.2).
3. Leverage publicly available vulnerability information (*e.g.* CVSS) via algorithm parameters so that the importance of security problems is computed with respect to vulnerability attributes such as attack complexity and exploit availability (Sections 3.4.1 and 3.4.2).

Contributions related to partial cuts

1. Introduce a directed graph representation of the relationship between closures (a closure relation graph). The closure relation graph gives insight into the AND/OR directed graph connectivity problem and enables the computation of multi-action COA recommendations that maximally disrupt connectivity, within a specified budget. Produce an implementation that builds and visualizes closure relation graphs. (Section 4.4.4).

2. Define the concept of the closure of a vertex set in the context of AND/OR directed graphs (Section 4.3).
3. Create and implement a polynomial-time best-first search algorithm that generates partial-cut vertex sets (COA recommendations). When integrated with the ranking algorithm, our approach combines knowledge of the logical structure of a network, vulnerability attributes, exploit maturity, asset prioritization, remediation costs, and budget limitations. Our approach enables the polynomial-time computation of partial-cut solutions which give favourable returns on investment (Section 4.4.5).

Contributions related to both ranking and partial cuts

1. Produce a prototype integrating our ranking and partial-cut vertex set algorithms with the MulVAL attack graph generator and a network discovery product. This serves to empirically validate our results through experiments and demonstrates that practical solutions can be found in polynomial-time for a representative single site corporate network. We also show that COAs for large enterprise networks can be computed in a reasonable time period (Section 5.2.4).
2. Create and implement a network generator which uses preferential attachment to produce simulated computer networks with services, vulnerabilities, attack goals, and attacker starting location (Section 4.2).
3. Recognize that AND/OR directed graphs and forward directed hypergraphs are equivalent, and create algorithms to convert between the two representations (Section 3.2).

Our work deepens the understanding of critical node identification together with connectivity-disruption in networks. In particular, we show the application to computer network security where the algorithms provide effective recommendations for improving security, even when practical considerations prevent the network from being completely secured.

1.5 Organization of thesis

Chapter 2 provides background on the differences between the types of graphs, and outlines related work. Chapter 3 introduces our graph ranking algorithm, AssetRank, explains its input parameters, and gives a stochastic interpretation of the results. Chapter 4 gives algorithms which compute partial-cut vertex sets containing key vertices whose deletion reduces connectivity (measured by the vertex AssetRanks) between two subgraphs. The vertex set is called a “course of action”. Chapter 5 contains experiments that evaluate the AssetRank and partial-cut algorithms. Chapter 6 summarizes the thesis, discusses ways the algorithms can be used, and the implication of using various parameters. It includes limitations of the current work, and suggestions for further work.

Chapter 2

Graph theory concepts

In this chapter we introduce graph theory concepts that will be used throughout the thesis. The discussion begins with the types of graphs and the attributes they have. Two matrix representations of graphs are presented, and spectral graph theory is introduced. In particular, its use in clustering and ranking graph vertices is examined. Networks with power-law degree distributions are reviewed and their generation is described. Finally, the semantics of attack graphs (fault graphs for computer network defence) are explored.

2.1 Types and attributes

The four types of graph are undirected graphs, directed graphs, undirected hypergraphs, and directed hypergraphs. Data is variously best suited to each of the different types and we give examples of each.

2.1.1 Undirected graphs

An undirected graph $G = (V, E)$ comprises of a set of vertices V and a set of edges E connecting the vertices. The elements of the edge set $E = \{\{u, v\} \mid u, v \in V\}$ are sets containing two vertices denoting that an edge exists between them.

An example of data well suited to an undirected graph representation is a social network. In this case the vertices represent people and the edges represent friendships between them. The data is well suited to an undirected graph because friendships are one-on-one and bidirectional by their nature. Figure 2.1 gives an example of an undirected graph for a social network. For the figure, people are represented by the set of vertices

$$V = \{a, b, c, d, e, f\}$$

and their friendships are represented by the set of edges

$$E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{a, f\}, \{b, c\}, \{c, e\}, \{d, e\}\} .$$

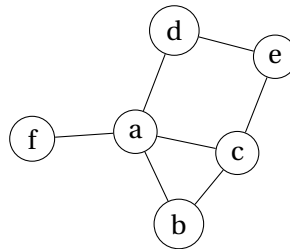


Figure 2.1: Example of an undirected graph — social network

2.1.2 Vertex and edge weights

Vertices and edges may both have attributes associated with them. Common attributes are labels, colours, and weights. Weights are assigned by a weighting func-

tion; in the case of edge weights, the function is called an affinity (or, similarity) function. Edge weights could reflect the number of connections between hosts in a network or the strength of affinities between neighbouring vertices while vertex weights could reflect production cost or profit. In this document, we define weight functions for graphs as follows.

Definition 2.1. A *vertex weight function* $f : V \Rightarrow \mathbb{R}^+$ assigns positive real number weights to vertices.

Weights are sometimes assembled into a vector for convenience. In this work, each vertex has two kinds of weight.

- **Cost:** The cost associated with a vertex v is denoted $c(v)$ and is a local property indicating the cost to remove the vertex or to initiate dissemination of a commodity at the vertex.
- **Rank:** The rank of a vertex v is denoted $r(v)$ and is a global property indicating the importance of the vertex in the graph.

Definition 2.2. An *edge weight function* $w : E \Rightarrow \mathbb{R}^+$ assigns positive real number weights to edges. For undirected and directed graphs the weights are often assembled into a square adjacency matrix that is indexed by the vertices.

- **Cost:** The cost associated with an edge e is denoted $c(e)$ and indicates the cost to remove the edge or to create the edge.
- **Affinity:** The affinity of an edge e , denoted $w(e)$, indicates the strength of the relationship represented by the edge.

2.1.3 Directed graphs

Directed graphs generalize undirected graphs by orienting the edges. The edges are called directed edges or arcs and the edge set is defined as $E = \{(u, v) \mid u, v \in V\}$. An example of data well suited to a directed graph representation is the World Wide Web. The data forms a “web graph” with web pages represented by vertices and links from one web page to another represented by arcs. The data is well suited to a directed graph representation because a hyperlink on page A to the page B does not imply that there will be a reciprocal link from page B to page A. Figure 2.2 gives an example of a web community represented by a directed graph. For the figure, pages are represented by the set of vertices

$$V = \{A, B, C, D, E, F\}$$

and hyperlinks between pages are represented by the set of ordered pairs

$$E = \{(A, C), (B, C), (C, D), (C, E), (C, F), (D, E), (F, C)\} .$$

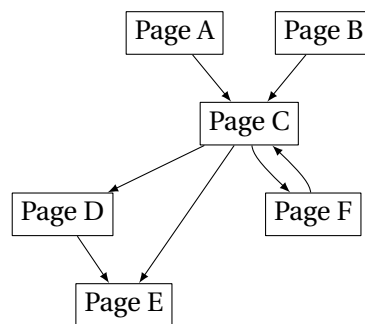


Figure 2.2: Example of a directed graph — web community

2.1.4 Hypergraphs

Hypergraphs are another generalization of undirected graphs. An undirected hypergraph relaxes the condition that each edge be comprised of exactly two vertices. While they do not represent more complicated sets of data than undirected graphs, they represent the data in a more compact fashion when the relations between data elements are not inherently one-to-one.

Hypergraphs connect multiple vertices with a hyperedge (a single undirected edge). A hypergraph $G = (V, E)$ is comprised of a set of vertices V and a set of edges E connecting the vertices. An element $e \in E$ is a set $e = \{v_1, v_2, \dots, v_k\}$ denoting that all of the vertices in e are connected by a single edge. If all of the edges contain exactly k vertices, the hypergraph is said to be k -uniform. A 2-uniform hypergraph is exactly an undirected graph.

An example of data well suited to a hypergraph representation is a set of travellers and the countries they have visited. Figure 2.3 is a hypergraph showing travellers, represented by vertices, and the countries they have visited, represented by hyperedges. For the figure, travellers are represented by the set of vertices

$$V = \{a, b, c, d, e\}$$

and countries are represented by the set of hyperedges

$$E = \{\{a, e\}, \{a, b, d, e\}, \{c, d\}\} .$$

Agarwal *et al.* [1] present two methods used to convert a hypergraph into an undirected graph. The first is a clique expansion algorithm. If $G = (V, E)$ is a hypergraph, the clique expansion algorithm creates a graph $G^c = (V^c, E^c)$ where $V^c = V$ and $E^c = \{\{u, v\} | u, v \in e, e \in E\}$. That is, the vertex set is unchanged and a complete

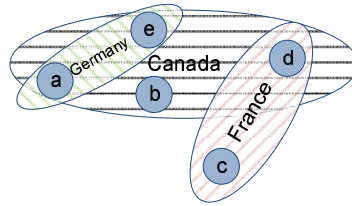


Figure 2.3: Example of a hypergraph — travellers

subgraph is added for each hyperedge. If the hyperedges are weighted, the edges may be weighted by the averaging function

$$w^c(u, v) = \arg \min_{w^c(u, v)} \sum_{\substack{e \in E \\ \{u, v\} \in E^c}} (w^c(u, v) - w(e))^2 . \quad (2.1)$$

The operator “argmin” in the above expression assigns a real number to $w^c(u, v)$ such that the summation is minimized.

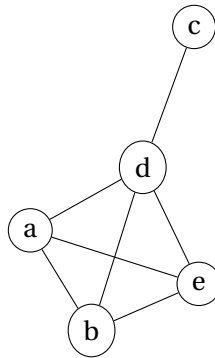


Figure 2.4: Clique expansion undirected graph of Figure 2.3

Figure 2.4 is a clique expansion undirected graph representation of the information in Figure 2.3. Note that information is lost with this representation and it is not possible to reconstruct Figure 2.3 from Figure 2.4. For example, it is impossible to know the countries the people have visited and, while a and e both went

to the same two countries, there is no stronger affinity between a and e than there is between a and b who have only visited a single common country.

Compare this with the star expansion algorithm that creates a bipartite graph $G^* = (V^*, E^*)$ by adding a new vertex to the set of vertices for every hyperedge, and adding edges from the new vertex to the vertices that were part of the hyperedge. Hyperedge weights are evenly distributed among all vertices in the edge. Formally,

$$V^* = V \cup E \quad (2.2)$$

$$E^* = \{(u, e) | u \in e, e \in E\} \quad (2.3)$$

$$w^*(u, e) = w(e)/d(e) \quad (2.4)$$

With this representation, the hypergraph can be reconstructed from the undirected graph. Figure 2.5 is a star expansion undirected graph representation of the information in Figure 2.3. Notice that no information is lost with the star expansion undirected graph but the representation is more complex.

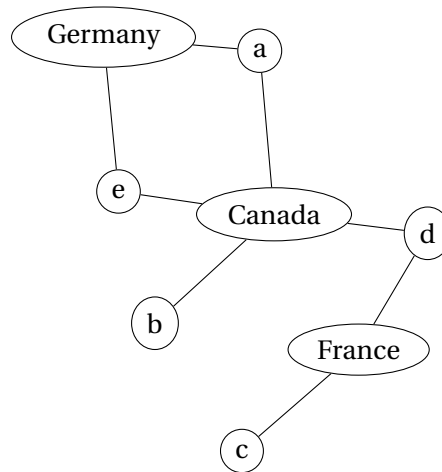


Figure 2.5: Undirected graph equivalent to Figure 2.3

2.1.5 Directed hypergraphs

The fourth type of graph, a directed hypergraph generalizes all of the preceding graphs. A directed hypergraph $G = (V, E)$ is composed of a set of vertices V and a set of edges E connecting the vertices. An element $e \in E$ is an ordered pair of disjoint vertex sets $e = (V^-, V^+)$ where the vertices in the set V^- are the sources of the directed hyperedge and the vertices in the set V^+ are the targets [28]. Edge weights are generalized so that an edge weight function $w : V \times E \Rightarrow \mathbb{R}$ assigns negative real numbers for each of the source vertices of an edge and positive real numbers for each of the target vertices of an edge.

Directed hyperedges can represent complex logical dependencies between the sources and targets of the edge.

Definition 2.3. An *F-arc* (*forward hyperarc*) is a directed hyperedge where the cardinality of the source vertex set is 1. That is, $|V^-| = 1$.

Definition 2.4. An *F-graph* (*forward hypergraph*) is a directed hypergraph whose edges are F-arcs.

Similarly, a B-graph is a directed hypergraph where $|V^+| = 1$ for all edges. An example of data well suited to an F-graph representation is an attack against computer networks. Attacker privileges are represented by vertices and the dependence upon preconditions that enable an attacker privilege are represented by F-arcs. Figure 2.6 gives an example of an attack graph represented by a directed hypergraph. For the figure, attacker privileges are represented by the set of vertices

$$V = \{a, b, c, d, e, g\}$$

and dependencies are represented by the set of directed hyperedges

$$E = \{(\{g\}, \{a\}), (\{g\}, \{b, c, d\}), (\{g\}, \{e\})\} .$$

The attack graph states that the goal is satisfied by obtaining a ; or e ; or all of b , c , and d .

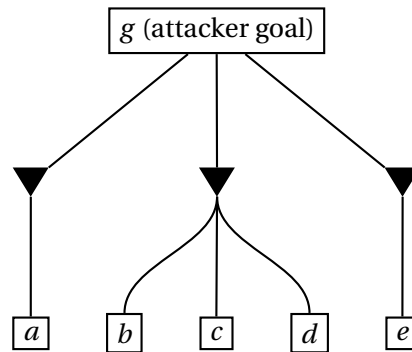


Figure 2.6: Example of a directed hypergraph — attack graph

Another common application of directed hypergraphs is Very Large Scale Integration (VLSI) circuits [6, 54]. VLSI circuits contain propositional logic formulas which translate to directed hyperedges with multiple source and target vertices. Following are two logic clauses:

$$a \wedge b \Rightarrow c \wedge d \tag{2.5}$$

$$a \vee b \Rightarrow e \wedge f \tag{2.6}$$

Those two logic clauses are encoded in the following directed hypergraph.

$$G = (V, E) \tag{2.7}$$

$$V = \{a, b, c, d, e, f\} \tag{2.8}$$

$$E = \{e_1, e_2, e_3\} = \{(\{a, b\}, \{c, d\}), (\{a\}, \{e, f\}), (\{b\}, \{e, f\})\} \tag{2.9}$$

We convert a directed hypergraph to a directed graph in a manner that is similar to the star expansion algorithm presented for hypergraphs. In order to preserve the meaning of edges as the graph is altered, the new expansion vertices that are added for the edges are assigned a type label of “AND”. The method is discussed in more detail in Section 2.1.6.

In summary, the four types of graphs can be arranged by the complexity of the relationships they represent. Undirected graphs represent the simplest relationships and, while hypergraphs do not represent more complex information than undirected graphs, they represent compound relationships in a more compact fashion. Directed graphs generalize both undirected graphs and hypergraphs, and directed hypergraphs represent compound directed relationships in a more compact fashion than directed graphs, although typed vertices are required to preserve meaning. In symbols: undirected graphs \equiv hypergraphs \subset directed graphs \equiv directed hypergraphs.

2.1.6 AND and OR vertices

As seen in Section 2.1.5, directed hypergraphs edges allow for a distinction between transitions from a vertex to *one* of its out-neighbours and a vertex to *all* of its out-neighbours. We apply a star expansion on an F-graph $G = (V, E)$, as described below and detailed in Algorithm 3.1, by adding vertices to represent its edges that have $|V^+| > 1$.

$$E' = \{e \mid e = (V^-, V^+) \in E, |V^+| > 1\} \quad (2.10)$$

$$V^* = V \cup E' \quad (2.11)$$

$$E^* = E \cup \{(V^-, \{e\}), (\{e\}, V^+) \mid e = (V^-, V^+) \in E'\} \quad (2.12)$$

$$G^* = (V^*, E^*) \quad (2.13)$$

The result is that no vertex in the F-graph has more than one F-arc with $|V^+| >$

1. The vertices can then be labelled with one of three types.

- AND vertex: New vertices added from E' ; each has a single out-arc with $|V^+| > 1$.
- OR vertex: Original vertices in V with out-arcs; every out-arc has the property $|V^+| = 1$
- SINK vertex: Vertices in V without out-arcs

Graphs with these vertex type restrictions can be visualized as directed graphs, by applying the “vertex type” attribute to each vertex. That is, a vertex where a transition may be made to any *one* of its out-neighbours is typed as an *OR* vertex and a vertex where a transition is made to *all* of its out-neighbours is typed as an *AND* vertex. Depending on the application, a vertex with no out-neighbours (a sink vertex) may be typed as an AND, OR, or SINK vertex since the type is often irrelevant.

Definition 2.5. A *vertex typing function* $h : V \Rightarrow \{AND, OR, SINK\}$ assigns a vertex label to vertices.

Exploit-dependency attack graphs are examples of an AND/OR directed graph. Figure 2.7 shows such an attack graph. OR vertices are represented with a diamond shape, AND vertices are represented with an ellipse shape and SINK vertices are represented with a box shape. The graph shows that the ability to execute arbitrary code on a PC can be obtained by a choice of two different means and hence the associated vertex is an OR vertex. One way to gain the ability is through a local login and the other is through a remote exploit. Both of the Local Login and Remote Exploit vertices are AND vertices since both of their out-neighbours are required in order to fulfill the attack. Note that directed hypergraphs are still more general than directed graphs with vertex types since directed hypergraphs allow multi-tail edges and a vertex can be the source of numerous multi-head edges.

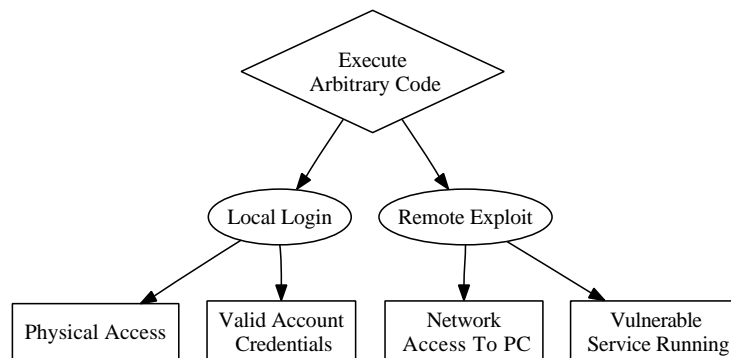


Figure 2.7: Example of AND/OR graph — attack graph

2.2 Matrix representations

Spectral graph theory applies the power of linear algebra to matrix representations of graphs and it is the study of their spectra. The spectrum of a matrix is technically defined as the set of eigenvalues of a matrix but in practice, the spectrum

also often refers to the eigenvalues, eigenvectors, and the characteristic polynomial. Graphs have two primary matrix representations, the affinity matrix and the Laplacian. The two representations may be derived from each other and two normalizations of the representations are common. The matrix used for the spectrum may be the affinity matrix or Laplacian matrix (and their normalizations). Since there is no standard choice, the matrix used will be explicitly stated. In this section we introduce the matrices associated with a graph and they are summarized in Table 2.1.

Name	Definition	Applicable to
Incidence/Boundary	B	All
Edge Weight Matrix	$W_e = \text{diag}(w(e_1), \dots, w(e_{ E }))$	Hypergraph
Affinity/Weight	$W_{ij} = w(i, j)$	Undirected, Directed
Affinity/Weight	$W = BW_e$	Hypergraph
Affinity/Weight	$W_{ve} = w(v, e)$	Directed Hypergraph
Probability Transition	$P = D^{-1}W$	Undirected, Directed
(Out) Degree	$D = \text{diag}(d_1, \dots, d_{ V }), d_i = \sum_{j=1}^{ V } w_{ij}$	Undirected, Directed
In Degree	$D^- = \text{diag}(d_i^-), d_i^- = \sum_{j=1}^{ V } w_{ji}$	Directed
Vertex Degree	$D = \text{diag}(d_1, \dots, d_{ V }), d_i = \sum_{e_j, v_i \in e_j} w(e_j)$	Hypergraph
Out Degree	$D^+ = \text{diag}(d_i^+), d_i^+ = \sum_{\substack{v_i \in V^+ \\ (V^-, V^+) \in e_j}} w(i, j)$	Directed Hypergraph
In Degree	$D^- = \text{diag}(d_i^-), d_i^- = \sum_{\substack{v_i \in V^- \\ (V^-, V^+) \in e_j}} w(i, j)$	Directed Hypergraph
Combinatorial Laplacian	$L = D - W = BB^T$	Undirected
Laplacian (Random Walk)	$L_{rw} = D^{-1}L = I - D^{-1}W$	Undirected
Laplacian (Symmetric)	$L_{sym} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}$	Undirected
Combinatorial Laplacian	$L = \Pi - \frac{\Pi P + P^T \Pi}{2}$	Directed
Laplacian (Symmetric)	$L_{sym} = I - \frac{\Phi^{1/2} P \Phi^{-1/2} + \Phi^{-1/2} P^T \Phi^{1/2}}{2}$	Directed

Table 2.1: Graph matrices

2.2.1 Affinity matrix

The affinity matrix W of an undirected or directed graph is a $|V| \times |V|$ matrix and is created by setting W_{uv} to $w(u, v)$ for each edge $(u, v) \in E$ and 0 otherwise. An adjacency matrix is a simplified affinity matrix that gives only connectivity information by setting W_{uv} to 1 if there is an edge from u to v , and 0 otherwise. For an undirected graph, W is symmetric, that is, $W = W^T$.

Example 2.1. The affinity matrix associated with the graph in Figure 2.1 is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.14)$$

Example 2.2. The affinity matrix associated with the graph in Figure 2.2 is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (2.15)$$

The incidence matrix B , is a $|V| \times |E|$ matrix where the rows correspond to vertices and the columns correspond to edges. The entries of B for undirected and directed graphs are (for undirected graphs, the edges are considered to be oriented but the choice of orientation does not matter):

$$B_{ij} = \begin{cases} -1 & \text{if } e_j = (v_i, w) \in E, \text{ for some } w \\ 1 & \text{if } e_j = (u, v_i) \in E, \text{ for some } u \\ 0 & \text{otherwise .} \end{cases} \quad (2.16)$$

For hypergraphs, $B_{ij} = 1$ if $v_i \in e_j$. In the case of directed hypergraphs, for each $v_i \in V$ and $e_j = (V_j^-, V_j^+) \in E$ the entries of the matrix are formed by

$$B_{ij} = \begin{cases} -1 & \text{if } v_i \in V_j^- \\ 1 & \text{if } v_i \in V_j^+ \\ 0 & \text{otherwise .} \end{cases} \quad (2.17)$$

The edge weights of a hypergraph may be assembled into a diagonal $|E| \times |E|$ weight matrix W_e . The affinity matrix of a hypergraph is given by $W = BW_e$.

Example 2.3. The affinity matrix associated with the hypergraph of travellers in Figure 2.3 is

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} . \quad (2.18)$$

For comparison, the affinity matrix of the equivalent undirected graph that was presented in Figure 2.5 follows.

Example 2.4. The affinity matrix associated with the equivalent undirected bipartite graph in Figure 2.5 is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (2.19)$$

The matrix in Example 2.4 is a good illustration of the extra dimensions that are present in many graph matrices. The graph is represented in 3 dimensions by the matrix in Equation 2.18 while the matrix in Equation 2.19 uses 8 dimensions to represent the same information.

Example 2.5. The affinity matrix for the directed hypergraph of the VLSI circuit in Equation 2.7 is

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \quad (2.20)$$

Other quantities we will need when working with graphs are vertex degrees and graph volume.

Definition 2.6. The *out-degree* of a vertex, denoted d_v^+ for a vertex v , is the sum of the weights on its outgoing edges; the *in-degree* is defined analogously. Formulas to calculate vertex degrees may be found in Table 2.1.

Definition 2.7. The *volume* of a graph $G = (V, E)$ is the sum of its vertex degrees. That is, $\text{vol } G = \sum_{i=1}^{|V|} d_i$.

2.2.2 Laplacian matrix

The Laplacian matrix is another matrix representation for graphs and for many problems it is the more natural representation to work with. The combinatorial Laplacian of an undirected graph is given by $L = D - W$ where D is the degree matrix and W is the affinity matrix. It may also be computed from the incidence matrix, $L = BB^T$.

There are two normalizations applied to affinity and Laplacian matrices, computed by multiplying by the inverse of the degree matrix. The first, the “random walk” normalization, creates a probability transition matrix P or a random walk Laplacian L_{rw} .

$$W_{rw} = P = D^{-1}W \qquad L_{rw} = D^{-1}L \qquad (2.21)$$

The second normalization preserves symmetry (if it exists) by multiplying both column i and row i of the unnormalized matrix by $1/\sqrt{d_i}$.

$$W_{sym} = D^{-1/2}WD^{-1/2} \qquad L_{sym} = D^{-1/2}LD^{-1/2} \qquad (2.22)$$

The normalized Laplacian of a directed graph is defined by Chung [17] to be

$$L_{sym} = I - \frac{\Pi^{1/2}P\Pi^{-1/2} + \Pi^{-1/2}P^T\Pi^{1/2}}{2} \qquad (2.23)$$

where I is the identity matrix, P is a probability transition matrix for the graph, Π is the diagonal matrix formed from the components of the vector π where π is the positive principal left eigenvector of P and π is normalized to sum to 1. This vector is in fact the unique stationary distribution of the matrix P . Chung's definition assumes that P has a unique positive principal eigenvector. Perron's theorem (see Meyer [60], for example) guarantees the existence of such a vector when P is irreducible and positive (equivalent to G being a strongly connected graph). A common strategy with web graphs is to force a strongly connected graph by setting all 0 entries in P to a small number $\epsilon > 0$.

The entries of π for an undirected graph G are shown [17] to be

$$\pi_v = \frac{d_v}{\sum_{u \in V} d_u} = \frac{d_v}{\text{vol } G} . \quad (2.24)$$

By substituting $\Pi = \frac{1}{\text{vol } G} D$, L_{sym} simplifies to $I - D^{-1/2} W D^{-1/2}$ so the normalized directed graph Laplacian is consistent with the definition of the symmetrically normalized Laplacian for undirected graphs.

Chung defines a combinatorial Laplacian of a directed graph to be

$$L = \Pi^{1/2} L_{sym} \Pi^{1/2} = \Pi - \frac{\Pi P + P^T \Pi}{2} . \quad (2.25)$$

For an undirected graph, this simplifies to $L = \frac{1}{\text{vol } G} (D - W)$. This definition differs from the combinatorial Laplacian of an undirected graph by the scalar $1/\text{vol } G$ and it requires the existence of a unique principal eigenvector, which is restrictive.

Agarwal *et al.* [1] make a significant contribution by showing that all work done to date on the application of Laplacians to hypergraphs is in fact equivalent to a conversion of the hypergraph to an undirected graph followed by the application of a combinatorial or normalized Laplacian.

In Section 2.1.4 we showed how a hypergraph can be converted to an undirected graph through a clique expansion and star expansion. Table 2.2 is a summary of the previous work on hypergraphs and Agarwal *et al.*'s demonstration of how their work is equivalent to spectral analysis of an undirected graph.

Algorithm	Year	Hypergraph expansion	Analysis matrix
Li	1996	Star	Adjacency
Gibson	1998	Clique	Adjacency
Bolla	1993	Clique	Combinatorial Laplacian
Rodriguez	2002, 2003	Clique	Combinatorial Laplacian
Zhou	2005	Star	Symmetrically-normalized Laplacian

Table 2.2: Hypergraph and undirected graph clustering are equivalent

Their conclusion is that while, on the surface, hypergraphs seem to capture more information about higher order relations, the same analysis can be handled using pairwise relations in undirected graphs. The conclusion is surprising because the higher order relations are not directly represented in undirected graphs but can only be detected with further analysis. However, using appropriate weighting functions, the semantics of the hyperedges are transferred to undirected graphs. The graph theoretic problems behind unsupervised and semi-supervised learning in hypergraphs can be analyzed using existing undirected graph techniques.

Agarwal *et al.* show that while Laplacians of higher order may be defined for hypergraphs, only the lowest order Laplacian, L_0 , can analyze vertices. The objects of L_0 are vertices and edges (termed 0-chains and 1-chains by algebraic topologists) but, for example, the higher order Laplacians, L_1 and L_2 , operate on edges,

triangles, and tetrahedrons (1-chains, 2-chains, and 3-chains) and so do not yield vertex functions useful for clustering, ranking, and other machine-learning problems.

At this time, the Laplacian of a directed hypergraph has not yet been discovered. Following are several examples of graph Laplacians.

Example 2.6. The Laplacian matrix associated with the social network graph in Figure 2.1 is

$$L = D - W = \begin{bmatrix} 4 & -1 & -1 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & 0 & -1 & 0 \\ -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & -1 & -1 & 2 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.26)$$

Example 2.7. To compute the normalized Laplacian of the directed graph in Figure 2.2, the weight $\epsilon = 0.001$ was added to the edge weights so that the graph is strongly connected. This guarantees that a (unique) positive eigenvector exists — something that is required by Chung's definition.

$$\begin{aligned}
L_{sym} &= I - \frac{\Pi^{\frac{1}{2}} P \Pi^{-\frac{1}{2}} + \Pi^{-\frac{1}{2}} P \Pi^{\frac{1}{2}}}{2} \\
&= \begin{bmatrix} 0.999 & -0.001 & -1. & -0.001 & -0.001 & -0.001 \\ -0.001 & 0.999 & -1. & -0.001 & -0.001 & -0.001 \\ -0.001 & -0.001 & 0.999 & -1. & -1. & -1. \\ -0.001 & -0.001 & -0.001 & 0.999 & -1. & -0.001 \\ -0.001 & -0.001 & -0.001 & -0.001 & 0.999 & -0.001 \\ -0.001 & -0.001 & -1. & -0.001 & -0.001 & 0.999 \end{bmatrix} \quad (2.27)
\end{aligned}$$

Example 2.8. We now consider the Laplacian matrix associated with the hypergraph in Figure 2.3. Bolla defines a Laplacian for unweighted hypergraphs using a degree matrix, edge matrix and incidence matrix of the graph [1]. Bolla's Laplacian is defined to be:

$$L = D_v - B W_e^{-1} B^T \quad (2.28)$$

which for this example is:

$$\begin{aligned}
 L &= \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} 5/4 & -1/4 & 0 & -1/4 & -3/4 \\ -1/4 & 3/4 & 0 & -1/4 & -1/4 \\ 0 & 0 & 1/2 & -1/2 & 0 \\ -1/4 & -1/4 & -1/2 & 1/4 & -1/4 \\ -3/4 & -1/4 & 0 & -1/4 & 5/4 \end{bmatrix} \tag{2.29}
 \end{aligned}$$

At this time, the Laplacian of a directed hypergraph has not been defined so it is not possible to present the Laplacian matrix for the graph in Figure 2.6.

2.3 Spectral graph theory

The spectrum of a matrix consists of its characteristic polynomial, eigenvalues, and eigenvectors. These are invariant for isomorphic graphs; however, two non-isomorphic graphs may still have an identical spectrum.

Definition 2.8. If the rank of a square matrix A is n , there are up to n unique eigenvalue-eigenvector pairs. The *eigenvalues* are the roots of the characteristic polynomial (the determinant of $A - \lambda I$) and they are paired with non-zero eigenvectors so that $\lambda_i X_i = AX_i$ and $\lambda_i \bar{X}_i = \bar{X}_i A$. X_i and \bar{X}_i are the *right-eigenvector* and *left-eigenvector* (respectively) corresponding to λ_i .

The two most developed applications of data mining using spectral graph theory techniques are vertex ranking and clustering. Vertex ranking assigns a quantitative measure of the prominence of objects in a system. For example, web search engines rank web pages using spectral methods so that user searches are very likely to return the most relevant pages. Clustering groups objects together with other objects that they are closely related to. For example, photo software uses spectral methods to detect subjects in images.

2.3.1 Edge cuts

Spectral graph theory has been successfully applied to clustering in all but directed hypergraphs. The goal of clustering is to partition vertices into groups, based upon their similarities, by removing edges. Good partitions are those that have low edge weights between the clusters and high edge weights within clusters.

Meilă and Shi [59] make the connection between the eigenvalues of the random walk Laplacian and the random walk adjacency matrix (the probability transition matrix $P = D^{-1}W$). Recall that W is the weighted affinity matrix, D the degree matrix, and $L_{rw} = I - P$ the random walk Laplacian of a graph. The eigenvectors of L_{rw} are the solutions to $L_{rw}X = \lambda X$. The following equations show that the eigenvectors of L_{rw} and P are directly related, where if λ is an eigenvalue of L_{rw} then $1 - \lambda$ is an eigenvalue of P .

$$L_{rw}X = \lambda X \tag{2.30}$$

$$\Rightarrow (I - P)X = \lambda X \tag{2.31}$$

$$\Rightarrow PX = (1 - \lambda)X \tag{2.32}$$

The same result is valid for the eigenvectors of the symmetrically normalized Laplacian and affinity matrices. These results are important because they show that after the normalization method is chosen, clustering using the Laplacian or the affinity matrix give the same result.

Although the eigenvectors of the normalized Laplacian and affinity matrices are the same, the clustering interpretation differs. The intuition of clustering the vertices of a random walk is to choose clusters that have a high probability of keeping the random walker transitioning between vertices in each cluster with a low probability of the random walker transitioning to another cluster. For the adjacency matrix, the eigenvectors of large eigenvalues correspond to areas of high connectivity and close semantic proximity. The eigenvectors of small eigenvalues correspond to noise or local characteristics. For the spectrum of the Laplacian, the opposite is true.

Partitions that are balanced in size are often desirable. The partitions may be balanced by the number of vertices in each cluster or the weight of the edges for the vertices in each cluster. These two cuts are referred to as the RatioCut [32] and NCut [78] (respectively).

Computing an optimal balanced cut is an NP-hard problem [84]. However, von Luxburg [83] gives an excellent explanation of how this discrete optimization problem can be relaxed into a real-valued optimization problem. The relaxed real-valued optimization problem is immediately solved by the k eigenvectors of the Laplacian matrix corresponding to the k smallest eigenvalues (where k is the number of clusters in the data). The real-valued solution must be converted back to a discrete solution and a common technique used for this purpose is k -means clustering [56]. Unfortunately, clusters obtained by the Laplacian + k -means method

do not necessarily bear any resemblance to the optimal solution [30].

The previous discussion relates to clustering of undirected graphs. Recently, definitions for cuts in directed graphs have been proposed [58, 90]. Clustering using the spectrum of directed graphs is more complicated since the affinity matrix is not symmetric and so the spectrum very likely contains complex numbers.

Meilă and Pentney [58] take a similar linear programming approach as was done in the undirected graph case. Finding an optimum clustering is NP-hard but the linear programming approach is to relax the integrality constraint on the indicator vector and find solution vectors Y_i whose elements are in the range $[0, 1]$. Meilă and Pentney's method gave very low classification errors on experiments with known correct clusters.

Agarwal *et al.* [1] prove that previous work on spectral clustering of hypergraphs reduces to first expanding the hypergraph to an undirected graph (with an appropriate weighting function) and then applying spectral graph theory in that realm. In Table 2.2 of Section 2.2.2, we gave the equivalencies between previous proposals for hypergraph clustering and their undirected graph equivalents. In the case of clique expansions, where the vertex set of the hypergraph and its expansion are the same, the eigenvectors of the hypergraph matrix and expansion graph matrix are identical; thus, the clustering of the undirected graph is the clustering of the hypergraph. In the case of star expansions, the vertex set is enlarged to $V^* = V \cup E$. If $X_{V^*} = [X_V \ X_E]$ is an eigenvector of a Laplacian or adjacency matrix of the expansion graph, then X_V is an eigenvector of the related hypergraph matrix with only minor algebraic manipulation.

Very little work has been done to date on clustering in directed hypergraphs and no work uses spectral methods. The most recent advance in partitioning di-

rected hypergraphs is Gallo *et al.* [26] where the minimum cardinality cut problem is reduced to finding a minimum set of hyperarcs covering the set of hyperpaths for the graph. They use a linear programming and divide-and-conquer approach and their experiments show that their technique has exponential complexity. Further, their work does not address directed hypergraphs in general but only B-graphs (and therefore also F-graphs by considering their transpose). The most recent work on cuts in general directed hypergraphs is by Liu *et al.* [54]. They give a gradient descent method of partitioning a directed hypergraph and their work does not involve spectral graph theory.

2.3.2 Vertex ranking

PageRank [67] and Hyperlink-Induced Topic Search (HITS) [46] are seminal algorithms that introduced vertex ranking for directed graphs in the context of web pages. Those algorithms, along with the Stochastic Approach for Link-Structure Analysis (SALSA) [50]; AuthRank, HubRank, and SymRank [20], are applications of spectral graph theory to directed graphs.

All of these methods use only the eigenvectors corresponding to the principal eigenvalue of their matrix. Kleinberg [46] shows several examples illustrating that the secondary (non-principal) eigenvectors sometimes show ranks within sub-communities in a directed graph; however, it is unknown which eigenvectors detect sub-communities, nor the subcommunity that is detected. Likewise, Skillicorn [79] gives several examples of how the secondary eigenvectors detect prominent vertices in sub-communities of an undirected graph. In his case, he uses the eigenvectors of $L_{rw}L_{rw}^T$ ¹ and plots the graph vertices using neighbouring

¹The paper uses the middle columns of the U matrix (the left singular vectors) of a singular value

eigenvectors, which gives a visual representation that can make it easier to identify vertices that might be prominent in sub-communities.

It is not known how to predict which eigenvectors will contain information on sub-communities, nor why one sub-community will be found while another is not. Borodin *et al.* [15] show that it is easy to construct examples where sub-communities exist but the secondary eigenvectors contain no information about them. Intuitively, the eigenvectors each correspond to an orthogonal vector corresponding to highly-correlated vertices so one expects they would in fact give information about sub-communities. To date, no rigorous results exist in this area.

PageRank [67] computes the stationary distribution of a probability transition matrix. The original graph is first modified, into what is referred to as a teleporting random walk, with the introduction of a random jump from each vertex with probability α . The random jump forces a strongly connected graph which guarantees the principal eigenvector is real, unique, and positive by Perron's theorem. Bianchini *et al.* [10] give a rigorous explanation of the random walker and stochastic process models which describe the semantics of PageRank for vertex ranking. The principal eigenvector gives the expected probability for a random walker to be found at each vertex. Computing the stationary distribution using the power method gives an intuitive model of a random walker.

It is known that the principal eigenvalue is always 1 for probability transition matrices. Haveliwala and Kamvar studied the second eigenvalue [33] and the condition number [43] of the PageRank affinity matrix. They give an upper-bound on the second eigenvalue which in turn gives the eigengap. The eigengap determines

decomposition of L_{rw} . The columns of U correspond to finding the eigenvectors of $L_{rw}L_{rw}^T = I - P - P^T + PP^T$. If P were symmetric (it is not, in general), the columns of U would be the eigenvectors of L_{rw} since the eigenvectors of L_{rw} and L_{rw}^2 are the same.

the speed of convergence when using an iterative approach to calculate the principal eigenvector. They show that the upper-bound is in fact $1 - \alpha$ and so is directly controllable by the algorithm user.

A matrix P is ill-conditioned if small perturbations produce relatively large changes in the matrix inverse. The condition number of P is a measure of the sensitivity of the matrix inverse to such perturbations, and it is defined to be the ratio of the norm of P over the norm of P^{-1} . Haveliwala and Kamvar show that the condition number of the PageRank matrix is $\frac{2-\alpha}{\alpha}$, showing that the PageRank matrix is stable under perturbations with the commonly used $\alpha = 0.15$ but becomes unstable as α approaches 0.

If the adjacency matrix of a graph is not irreducible, the principal eigenvector is not guaranteed to be unique [23, 48]. For example, HITS and SALSA are two graph mining algorithms where the representation matrix is not irreducible. Thus there will be as many principal eigenvectors as the multiplicity of the largest eigenvalue (in absolute value). These algorithms calculate their output using an iterative power method and so they will return one of the principal eigenvectors but the actual vector computed is sensitive to the initialization vector's value.

2.3.3 Link prediction

In many networks, it is interesting to predict the future interactions that will occur. For citation networks this could predict which authors will co-author a paper together and in terrorist cells, link prediction could discover individuals who are interacting but have not been detected yet. Liben-Nowell and Kleinberg [53] obtained good results by applying the PageRank algorithm to a citation graph. They tested five data sets from the physics e-Print arXiv where their problem was to take

the data sets from 1996 and try to predict the co-authorings that would occur. It is not possible to know all future collaborations until the individuals in the data set have ceased working but they did look at which co-authorings had occurred by 1999.

In all five data sets, PageRank exceeded the prediction performance of simple graph distance. In three of the five data sets, it exceeded the performance of common neighbours prediction. However, the performance greatly depended upon the random jump parameter (damping factor) of PageRank and was only better for some values. The damping factor values they tested were 0.01, 0.05, 0.15, 0.30, and 0.50. The optimal values across the five data sets were not consistent and essentially spanned the extreme values with optimal values ranging from 0.05 to 0.50. Research is needed to discover what parameter values and edge weights are suitable, and under which conditions.

2.4 Power-law degree distributions

Many networks of real-world interest have vertex degrees that asymptotically exhibit a power-law distribution. In particular, Internet router connectivity, World Wide Web (WWW), biological cells, contagion spread, and social networks exhibit power-law distributions. In directed graphs, where we are interested in both the in-degree and out-degree of the vertices, the degree distributions have power law tails:

$$P_{out}(k) \sim k^{-\gamma_{out}} \text{ and } P_{in}(k) \sim k^{-\gamma_{in}} . \quad (2.33)$$

Here, $P_{out}(k)$ and $P_{in}(k)$ give the probability that a randomly selected vertex will have an out-degree or in-degree (respectively) of k ; and, $-\gamma_{out}$ and $-\gamma_{in}$ give the

slopes of the distributions in a doubly logarithmic plot.

Barabási and Albert [9] introduce a model for what they term “scale-free networks”; they later give a thorough survey of networks including analytical tools, mathematical properties, models, and applications [2]. They claim [9, p. 510] that preferential attachment and growth “are responsible for the power-law scaling observed in real networks.” Further, they claim that networks like the Internet are highly resilient to random node (router) removal but highly susceptible to attacks that remove the most connected nodes [8]. However, Doyle *et al.* [21] show that while Internet router connectivity follows a power-law distribution, it was not formed through preferential attachment, nor is it scale-free according to Albert and Barabási’s definition. Further, Li *et al.* [52] address the imprecision and contradictions in the various definitions of “scale-free” that are found in the literature. They introduce a metric that gives a measure indicating the extent to which a graph is scale-free. The work considers undirected graphs but not directed graphs.

2.5 Decision support for computer network defence

2.5.1 Attack graph semantics

There are basically two types of attack graphs. In the first type, each vertex represents the *entire* network state and the arcs represent state transitions caused by an attacker’s actions. Examples are Sheyner’s scenario graph based on model checking [75], and the attack graph in Swiler and Phillips’ work [81]. This type of attack graph is sometimes called a *state enumeration attack graph* [63]. In the second type of attack graph, a vertex does not represent the entire state of a system but rather a system condition in some form of logical sentence. The arcs in these

graphs represent the causality relations between the system conditions. We call this type of attack graph a *dependency attack graph*. Examples are the graph structure used by Ammann *et al.* [4], the *exploit dependency graphs* defined by Noel *et al.* [63, 64], the MulVAL *logical attack graph* by Ou *et al.* [65], and the *multiple-prerequisite graphs* by Ingols *et al.* [39].

The key difference between the two types of attack graphs lies in the semantics of their vertices. While each vertex in a state enumeration attack graph encodes all the conditions in the network, a vertex in a dependency attack graph encodes a single attack asset of the network. A path $s_1 \rightarrow s_2 \rightarrow s_3$ in a state enumeration attack graph means that the system's state can be transitioned from s_1 to s_2 and then to s_3 by an attacker. But the condition that enables the transition $s_2 \rightarrow s_3$ may have already become true in a previous state, say s_1 . The reason the attacker can get to state s_3 is encoded in some state variables in s_2 , but the arcs in the graph do not directly show where these conditions were first enabled. In a dependency attack graph, however, the dependency relations among various assets are directly represented by the arcs. In this document, we focus on dependency attack graphs because they are F-graphs that we can readily generate, and they have a direct application to computer network security.

2.5.2 Attack graph ranking

Mehta *et al.* apply the Google PageRank algorithm to state enumeration attack graphs [57] and claim to produce a rank that can serve as a metric for a system's overall vulnerability. The source vertex used in their application of PageRank is the network initial state. The rank computed represents the probability that a random attacker (similar to the random walker in the PageRank model) is in a specific state,

in particular, a state where he has achieved his goal. However, the probability a random attacker is in the goal state often decreases as the number of attack paths increases — simply because there are more states to split the distribution. As a result, this rank cannot serve as a metric for the system’s overall vulnerability.

Recent years have seen a number of efforts that apply numeric security metrics to attack graphs. For example, Wang *et al.* studied how to combine individual security metrics to compute an overall security metric using attack graphs [86]. Dewri *et al.* proposed configuration optimization methods that are based on attack graphs, numeric cost functions, and genetic algorithms [19]. The goal of our work is different. We aim to use standardized security metrics and a unified algorithmic framework to rank and prioritize the security problems revealed by an attack graph.

2.5.3 Graph cuts and course of action decision support

There are some established concepts in graph theory that we build upon. A k -connected graph is a graph where k is the least number of vertices whose deletion will disconnect the graph. Local vertex connectivity of vertices u and v gives the least number of vertices that will disconnect u and v . We extend these terms to consider the optimal set of vertices, whose cost does not exceed k , that will maximally (though not necessarily completely) disrupt the connectivity between a source and target set in a ranked graph.

AND/OR directed graphs correspond to propositional logic formulas and the SINK vertices are logic variables that may be true or false. When an attack graph is built, all SINK vertices are set to true (representing the fact that vulnerabilities exist, certain software is installed, network paths exist, *etc.*). The remaining ver-

tices in the graph build conjunctive (AND vertices) and disjunctive (OR vertices) clauses. In addition, each vertex corresponds to a logical statement that can be simplified to consist entirely of SINK vertices. The statement will be true or false, depending upon the value of the variables (the SINK vertices). Wang *et al.* [85] compute optimal COAs by building logic formulas for the goal vertices. However, they point out that building the formulas requires exponential time, and so a different approach is required to be useful in practice. Furthermore, if setting one or more SINK vertices to false does not falsify the goal, the approaches in previous work [36, 85] will not indicate the degree to which the security of the network has changed. The approach is related to vertex cut-sets since a solution that falsifies the goal is also a vertex cut-set between the goal and attacker.

A number of previous papers have studied the problem of computing course-of-action recommendations for IT networks. Jha *et al.* [42] uses attack graphs to deduce a minimal set of security measures that will ensure security. Our work differs insofar as we maximize the value of privileges denied to the attacker, but within a fixed budget. We are interested in solutions that will measurably, although not necessarily completely, improve the security of the system for a given cost. Ingols *et al.* [39] build “multiple prerequisite” attack graphs, and generate single-action recommendations based on the number of hosts secured by deleting vulnerability instances. Our work extends their work by considering costs and combinations of actions, as well as the relative value of privileges to the defender. Jajodia *et al.* [68] propose a “weakest-adversary” metric in which the security of a network is stated in terms of the weakest adversary that can successfully attack it. Their approach permits a comparison of the relative security of two different network configurations.

Dewri *et al.* also study the problem of how to select a set of security hardening measures that satisfy a budgetary constraint [19]. They use genetic algorithms to search for a solution to a multi-objective optimization problem that balances security improvements against cost and potential damage. Their work deals with attack trees while our work deals with the more general case of attack graphs. Recall that any two vertices in a tree are connected by exactly one path, while in a graph they may be connected by multiple and cyclic paths.

Directed graph theory defines an (x, y) -vertex cut-set as a set of vertices whose removal disconnects vertices x and y (for example, representing an attacker and goal in a network attack graph). Directed graph cut-set research is not directly applicable to the problem addressed in this thesis for two reasons. The first, and most important reason, is that we do not assume it is possible to remove all connectivity between attackers and goals. Second, logic attack graphs are not properly represented by standard directed graphs since they are in fact F-graphs, a kind of directed hypergraph. We are not aware of any published theory on vertex cuts in directed hypergraphs.

Wang *et al.* [85] use a minimum-cost algorithm to identify sets of network properties that enable successful attacks. They associate a propositional logic formula with each attack graph vertex that states its truth condition as a function of the truth values of network configuration facts. In exponential time, they search for minimal-cost sets of facts that can invalidate the attack.

Homer and Ou [35] combine MulVAL-generated attack graphs with usability requirements, and use Boolean Satisfiability Solving to find network configurations that provide security while retaining usability. They compute the minimal cost cut-set that will completely protect a set of identified vertices while respect-

ing specified usability requirements. If a solution exists, their method provably finds the optimal (lowest cost) configuration. The attempt to find solutions that simultaneously address security problems and preserve usability is admirable. The advantage of our work is that it can provide an effective course of action even when absolute protection of the goal vertices is not possible.

Our work extends existing approaches by employing rank weights, which permit strategic defence of network assets that are more valuable to the attacker. We also produce partial solutions when mitigation costs of a complete solution are too high. If each vertex is assumed to be of equal value to the attacker and the budget is infinite, our brute-force exponential-complexity solution is similar to the previous exponential approaches.

Borgatti [14] defines the problem of finding sets of key players in social networks. The negative key player problem (KPP-Neg) is finding a set of players whose removal results in a network with the least possible cohesion (*i.e.*, their removal maximally disrupts the network).

Borgatti's work generalizes Friedkin's centrality measure work [25]. Friedkin refined previous centrality measures [12,13,16,18,24,38,44] (which were seen to be competing) into a unified, complimentary framework. He defined three centrality measures:

- Total Effects Centrality (TEC): indicates the total relative effect of an actor on the other actors of the network. The TEC measures correspond to the principal eigenvector of the network's influence matrix.
- Immediate Effects Centrality (IEC): indicates the immediacy of an actor's total effects

- Mediative Effects Centrality (MEC): indicates the extent to which an actor mediates the total effects of other actors

Friedkin's measures are computed for *each* actor in a network. With KPP-Neg, Borgatti would like to compute a set that (together) has the maximal mediative effect (Friedkin's MEC) on the network.

Finding an optimum set for KPP-Neg does not translate into merely selecting the top n network objects with respect to MEC measures. For example, a maximally disruptive KPP-Neg set of size 2 need not contain the vertex in the KPP-Neg set of size 1. Borgatti's work was done in edge-weighted undirected graphs while our application area is F-graphs with cost and rank vertex weights. The cost weight indicates the cost to remove the vertex and the rank weight indicates the influence of the vertex in the graph.

2.6 Supermodular knapsack problem

The optimization problem we will give in Section 4.4 is a formulation of the supermodular knapsack problem, which is part of a class of problems known as nonlinear knapsack problems. Intuitively, the nonlinear knapsack problem considers how to fill a knapsack with items where the space of the knapsack is fixed, each item takes up some space, and each item yields a profit. However, the profit of combinations of items is more than the simple sum of the profits of the items individually. The question then is how to fill the knapsack with items that yield the maximum profit.

The nonlinear knapsack problem is specified by Gallo *et al.* [29] as given below. This type of knapsack problem is also known as an unbounded multiply-

constrained (or, multidimensional) knapsack problem.

$$\max Q(C) \text{ where } Q : \{0, 1\}^n \Rightarrow \mathbb{R} \quad (2.34)$$

under the constraint

$$K \cdot C \leq b \quad (2.35)$$

where K is a positive real vector, $C \in \{0, 1\}^n$ is an indicator vector, $\sum K_i > b > 0$, $K \cdot C$ denotes the scalar product of K and C . Also, $Q(\mathbf{0}) = 0$, $Q(\mathbf{1})$ is the unique maximum point of $Q(C)$. The problem can be extended by allowing additional constraints. The supermodular case occurs when Q is supermodular:

$$Q(X \wedge Y) + Q(X \vee Y) \geq Q(X) + Q(Y), \text{ for all } X, Y \in \{0, 1\}^n \quad (2.36)$$

Relating to Section 4.3, the evaluation function Q is the sum of ranks of the closure of a set of vertices to remove. That is, Q takes an indicator vector of vertices to remove from the graph, computes the closure of those vertices in the context of source and target vertices, and the resulting value is the sum of the ranks of the vertices in the closure. The function is supermodular because the closure of the union of two sets of vertices always contains at least the union of the closure of each separate set. We have no reason to believe that our function computing rank sums of closures is representable by an easily computed polynomial. Thus, it is treated as an oracle for which we know the computational complexity. Relating back to the intuitive description of the knapsack problem, the profit in our case is the sum of ranks of the closure, the knapsack space taken by each item is the cost of removing the vertex from the graph, and size of the knapsack is the total budget available when removing vertices.

Applications involving the nonlinear knapsack problem arise often and the area has had active research over the last 30 years (for example, [5,27,40,49,80,89]). Gallo *et al.* [29] investigated relaxations of the problem (*i.e.*, relaxing the domain restriction on the indicator vector components from $\{0,1\}$ by allowing any real number in $[0,1]$). The relaxation can then be solved in polynomial time. This relaxation is the same approach as the one taken in computing cut-sets using the Lagrangian matrix of a graph. Although their results are theoretically interesting, they are not practical in our problem because our evaluation function is not given as a polynomial and so computing its relaxation would require computing all points in its domain. That is, computing the function's relaxation is at least as difficult as computing COAs.

The confusion of terminology makes the literature sometimes challenging to review. For example, a recent publication [51] claims to “guarantee the find of an optimal exact solution to the primal problem” for nonlinear knapsacks. However, their definition of the nonlinear knapsack problem is the second kind that is sometimes seen in the literature. They define the quantity to be maximized as:

$$\max f(X) = \sum_{j=1}^n f_j(X_j) \quad (2.37)$$

where $X \in \mathbb{Z}^n$. However, notice that this function definition only considers one component of the vector X at a time (and thus is not supermodular) and applies a different function to each component.

Chapter 3

Vertex ranking

In Section 2.1.6 we discussed the semantics of AND/OR digraphs and their correspondence to F-graphs. Computer network security is a common source of AND/OR digraphs corresponding to attacks. In this section we frame the discussion of F-graph ranking around computer network attack graphs in order to apply the concepts and make their discussion clearer. However, the work is general and may be applied to AND/OR digraphs arising from any source.

3.1 Introduction

An attack graph is a mathematical abstraction of the details of possible attacks against a specific network. Various forms of attack graphs have been proposed for analyzing the security of enterprise networks [4, 39, 64, 65, 69, 76]. Advances have enabled computing attack graphs for networks with thousands of machines [39, 65]. Even when attack graphs can be efficiently computed, the resulting size and complexity of the graphs is still too large for a human to fully comprehend [62,

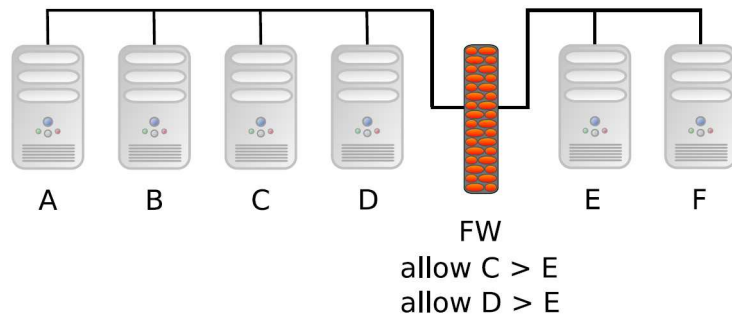


Figure 3.1: An example network from Ingols *et al.* [39]

63, 87]. While a user will quickly understand that attackers can penetrate the network, it is practically impossible to know which privileges and vulnerabilities are the most important to the attackers' success. Network administrators require a tool that can distill the overwhelming amount of information into a list of priorities that will help them to secure the network, making efficient use of scarce human and financial resources.

The problem of information overload can occur even for small-sized networks. In the example network of Figure 3.1, Machine A is an attacker's launch pad (for example, the Internet). Machines B, C, and D are located in the left subnet and machines E and F are in the right subnet. The firewall FW controls the network traffic such that the only allowed network access between the subnets is from C and D to E. All of the machines have a remotely exploitable vulnerability.

We applied the MulVAL attack graph tool suite [65] to the example network. The resulting attack graph can be found in Figure 3.2. Even for a small network, the attack graph is barely readable on a full page. Assuming the attack graph can be read, it is still difficult for a human to capture the core security problems in the simple network. Essentially, the software vulnerabilities on hosts C and D will

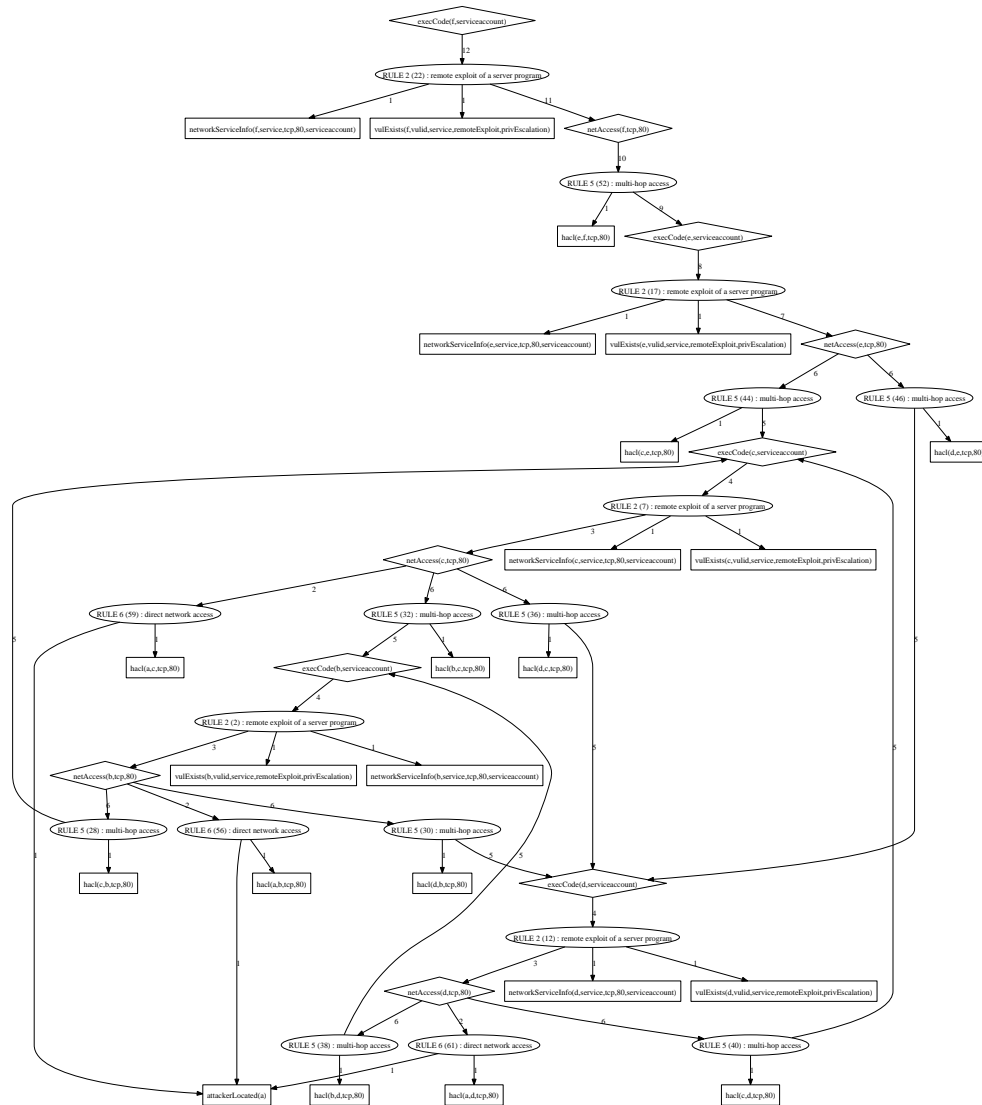


Figure 3.2: Attack graph for the network in Figure 3.1

enable an attacker from A to gain local privileges on the victim machines, and use them as stepping stones to penetrate the firewall, which only allows through traffic from C and D. In this example, all the machines can potentially be compromised by the attacker, and all the vulnerabilities on the hosts can play a role in those potential attack paths. However, the vulnerabilities on C and D, and the potential compromise of those two machines, are crucial for the attacker to successfully penetrate into the right subnet, presumably a more sensitive zone. The attack graph produced by MulVAL does reflect this dependency, but a careful reading of the graph is necessary to understand which graph vertices are the most important to consider. When the network size grows and attack paths become more complicated, it is insurmountably difficult for a human to digest all the dependency relations in the attack graph and identify key problems.

Besides the dependency relations represented in an attack graph, another important factor in determining the criticality of an identified security problem is the likelihood that the attack path can lead to a successful exploit. For example, both hosts C and D can be exploited remotely by the attacker on host A. Assume that the vulnerability on host C is only theoretical and no one has successfully produced a proof-of-concept exploit, whereas the vulnerability on host D has a publicly available exploit that works most of the time. Obviously the vulnerability on D is more likely to be exploited than the vulnerability on C and so its elimination deserves prioritization.

In the past decade, significant resources have gone into standardizing the definition of the attributes of reported security vulnerabilities. Most notably, the Common Vulnerability Scoring System (CVSS)¹ is a standard for sharing the at-

¹<http://www.first.org/cvss/>

tributes of discovered security vulnerabilities among IT security professionals. It represents not just a single numeric score, but a metric vector that describes various aspects of a vulnerability such as its access vector, access complexity and exploitability. The CVSS metric vector is included in the National Vulnerability Database (NVD)² for every vulnerability reported in NVD. The metrics provide crucial baseline information for automated security analysis. However, the metrics themselves can only give limited information without an understanding of the global security interactions in an enterprise environment. For example, further assume that the vulnerability on B is the same as the one on D. Since B does not have access into the right subnet, its vulnerability is less critical than the one on D. In the scenario just described, our algorithm gives first priority to the vulnerability on D, followed by the vulnerability on B, and then C. This prioritization is intuitive since D is easy to exploit and gives access to the right subnet; B is easy to exploit and gives access to D; and since only proof-of-concept code exists to exploit C, it warrants the lower priority. All of the parameters in our ranking algorithm can be tuned to model attackers of various levels of sophistication and technique.

In order to determine the relative importance of security problems in a network, both the dependency relationships in the attack graph *and* the attributes of the security problems need to be considered. We present an approach that automatically digests the dependency relations in an attack graph as well as the baseline information of the vulnerability attributes to compute the relative importance of attacker assets (the graph vertices) as a numeric metric. The metric gauges the importance of a privilege or vulnerability to an attacker (and hence the defender). Our approach fuses attack graphs and baseline security metrics such as CVSS, to

²<http://nvd.nist.gov/cvss.cfm>

make both of them more useful in security analysis. The product is primarily a defensive tool that gives the advantage to network defenders since they have full information about their network and can build a complete attack graph whereas attackers will usually have incomplete information.

Our algorithm is based on spectral methods and is similar to the Google PageRank algorithm [67] which ranks the importance of web pages. It is important to note that our work is significantly different from previous work in applying PageRank to state enumeration attack graphs [57].

First, our work extends the original PageRank algorithm by generalizing its damping factor and providing the ability to operate on directed F-graphs. Our work shows how our PageRank generalizations, the dependency matrix, and personalization vector can be set to obtain rich security insight from the fusion of attack graphs with attack asset attributes, such as the maturity of exploit code. Our extended PageRank algorithm is named AssetRank.

Second, we have approached the problem using dependency attack graphs which have very different semantics from the state-enumeration attack graphs used in the previous work (see Section 2.5.1). The interpretation of the computed rank values are completely different for different graph semantics and it is important to understand what the values mean in any new context.

Dependency attack graphs are AND/OR directed graphs. The metric the AssetRank algorithm computes indicates the value of an attack asset (a graph vertex) to a potential attacker. Attack assets consist of privileges, such as the ability to execute code on a particular machine, and facts, such as the existence of vulnerable software on a host. We give a stochastic interpretation of the asset ranks in the context of network attacks and conduct experiments on various network settings.

The results of our experiments show that the vertex ranks computed by our algorithm are consistent, from a security point of view, with the relative importance of the attack assets to an attacker. Asset ranks computed on attack graphs that have been fused with isolated metrics from CVSS data will provide a contextualized understanding of the importance of network facts, based on the network interdependencies. The asset ranks of vulnerabilities in various networks could add value to the CVSS metrics by giving an understanding of the significance of vulnerabilities across diverse environments. The asset ranks can be used to prioritize countermeasures, help a human reader to better comprehend security problems, and provide input to further security analysis tools, including our course of action algorithms presented in Chapter 4.

3.2 Conversion between F-graphs and AND/OR digraphs

F-graphs and AND/OR graphs are logically equivalent although an AND/OR graph could require the introduction of new vertices in order to represent the data in an F-graph. We created Algorithms 3.1 and 3.2 to convert between F-graphs and AND/OR graphs. Performing a round-trip conversion will result in a graph that is logically equivalent but it may lengthen paths because extra vertices are introduced in Algorithm 3.1. A more sophisticated version of Algorithm 3.2 would produce a minimal F-graph that is logically equivalent the AND/OR directed graph.

3.3 AssetRank for AND/OR digraphs

A dependency attack graph G is represented as $G = (V, A, f, g, h)$ where V is a set of vertices; A is a set of arcs represented as (u, v) , meaning that vertex u depends on

Algorithm 3.1 Convert F-graph to AND/OR directed graph

Input: F-graph $G = (V, A)$ **Output:** G' , an AND/OR directed graph representation of G

```

1:  $V' \leftarrow \emptyset, A' \leftarrow \emptyset, G' \leftarrow (V', A')$  {Initialize directed graph  $G'$ }
2: for  $v \in V$  do
3:   Add  $v$  to  $V'$  with a type of OR
4:   for every out-arc of  $v$ :  $(\{v\}, U) \in A$  do
5:     for every  $u \in U$  do
6:       Add  $u$  to  $V'$  and set the type to OR
7:       if  $|U| = 1$  then
8:         Add an arc from  $v$  to the vertex in  $U$ 
9:       else  $\{|U| > 1\}$ 
10:        Add a new AND vertex  $v_U$  to  $V'$ 
11:        Add an arc from  $v$  to  $v_U$ 
12:        Add an arc from  $v_U$  to every vertex in  $U$ 
13: return  $G'$ 

```

Algorithm 3.2 Convert AND/OR directed graph to F-graph

Input: AND/OR directed graph $G = (V, A)$ **Output:** G' , an F-graph representation of G

```

1:  $V' \leftarrow \emptyset, A' \leftarrow \emptyset, G' \leftarrow (V', A')$  {Initialize F-graph  $G'$ }
2: for  $v \in V$  do
3:   Add  $v$  to  $V'$ 
4:   if the type of  $v$  is AND then
5:     Let  $U$  be the set of out-neighbours of  $v$ 
6:     Add the vertices  $U$  to  $V'$ 
7:     Add the hyperarc  $(\{v\}, U)$  to  $E'$ 
8:   else if the type of  $v$  is OR then
9:     for each out-neighbour  $u$  of  $v$  do
10:      Add  $u$  to  $V'$ 
11:      Add the hyperarc  $(\{v\}, \{u\})$  to  $A'$ 
12: return  $G'$ 

```

vertex v ; f is a mapping of positive weights to vertices; g is a mapping of nonnegative weights to arcs; and h is a mapping of vertices to their type (AND, OR, or SINK). The *out-neighbourhood* of a vertex v is defined as $N^+(v) = \{w \in V : (v, w) \in A\}$, and *in-neighbourhood* of v is defined as $N^-(v) = \{u \in V : (u, v) \in A\}$. The cardinality of a set X is denoted $|X|$ and its L1-norm is denoted $\|X\|_1$. Without loss of generality, we require the vector of all vertex weights $f(V)$ to sum to 1.

AssetRank is computed by solving for the principal eigenvector X in the following equation:

$$\lambda X = (D\Delta + \gamma P e^T) X \quad (3.1)$$

where λ is the principal eigenvalue, X is the vector of AssetRanks (scaled to sum to 1), D is the transpose of the square affinity matrix of a dependency attack graph G (an AND/OR directed graph), Δ is a diagonal matrix of vertex-specific arc-weight damping factors where each value is in the range $[0, 1]$, $\gamma \in (0, 1]$ is the vertex-weight damping factor, $P = f(V)$ is a personalization vector composed of the vertices' personalization values (that is, the vertex weights), and e is the all-ones vector. We will show the existence of a unique solution using Perron's theorem which requires the matrix $D\Delta + \gamma P e^T$ to be positive. Since D, Δ , and γ are (at least) non-negative, $D\Delta + \gamma P e^T$ is guaranteed to be positive if P is positive.

Equation (3.1) reduces to the original PageRank if $\lambda = 1$, $\Delta = \delta I$ (where I is the identity matrix and δ is PageRank's damping factor), $\gamma = 1 - \delta$, and all vertices are required to be OR vertices.

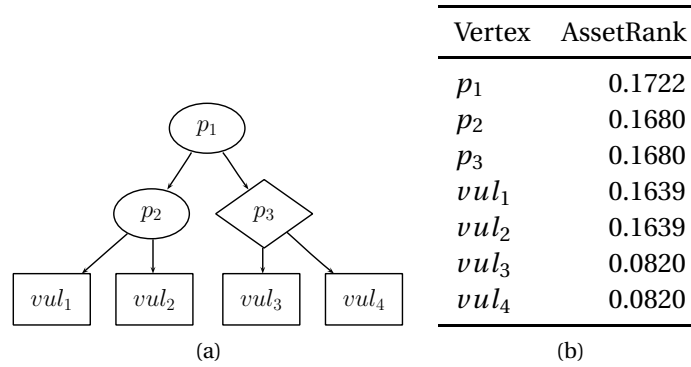


Figure 3.3: AssetRank computation for an AND/OR graph. In our figures, AND vertices are represented by ovals, OR vertices by diamonds, and SINK vertices by rectangles.

3.3.1 AND vertices

The example dependency attack graph in Figure 3.3a shows that attackers attaining the goal p_1 depend upon their ability to obtain both privileges p_2 and p_3 . p_2 is an AND vertex and it requires the two vulnerabilities vul_1 and vul_2 . p_3 is an OR vertex and it requires only one of either vul_3 or vul_4 . In this example we assume all the arcs have the same weight.

Since any of an OR vertex's out-neighbours can enable it, the importance of each out-neighbour decreases as the number of out-neighbours increases since the vertex can be satisfied by any one of them. This reduced dependency is not true of AND vertices. Since all the out-neighbours of an AND vertex are necessary to enable it, it is intuitively incorrect to lessen the amount of value flowed to each out-neighbour as their numbers grow.

Rather than splitting the value of an AND vertex we *replicate* it to its out-neighbours. Each out-neighbour of an AND vertex receives a copy of the full damped

value from the vertex. That is, for every outgoing edge (u, v) from an AND vertex u , the corresponding matrix entry D_{vu} ³ is 1. We now have the following restrictions on the graph's arc weights:

$$\sum_{w \in N^+(v)} g(v, w) = \begin{cases} |N^+(v)|, & \text{if } h(v) = \text{AND} \\ 1, & \text{if } h(v) = \text{OR} \\ 0, & \text{if } h(v) = \text{SINK} \end{cases} \quad (3.2)$$

3.3.2 Convergence

We first need to show that the principal eigenvector X in Equation (3.1) exists (up to scalar multiplication) and is unique. Then, we will show how to compute the solution using the power method. The facts of existence and uniqueness come from Perron's Theorem which gives the following facts for a square matrix with all positive elements (see, for example, [60]):

1. A positive eigenvalue λ exists;
2. All other eigenvalues (possibly complex) are less than λ in absolute value (thus, λ is the unique principal eigenvalue);
3. Since λ is unique (has multiplicity one), there is one corresponding eigenvector (up to scalar multiplication);
4. All elements of the principal eigenvector are positive; and
5. It is the only eigenvector whose entries are all positive.

³As a shorthand notation we use u and v in D_{vu} to represent the column and row indices corresponding to the respective vertices.

Note that $D\Delta + \gamma P e^T$ is positive by the conditions on the individual components, therefore Perron's Theorem applies and so the principal eigenvector X exists, is unique, and is positive. The Perron-Frobenius Theorem extends Perron's Theorem and gives the above facts for nonnegative (*i.e.*, zero entries are allowed) matrices that are irreducible. However, as discussed in Section 2.3.2, if the matrix is reducible, then the principal eigenvalue can have multiple nonequivalent eigenvectors, as Farahat *et al.* [23] show for the HITS and SALSA ranking algorithms.

The power method is a simple computational algorithm for computing an approximation to the principal eigenvalue and eigenvector of a matrix. The conditions for it to converge to a solution are:

1. The matrix has a unique principal eigenvalue; and
2. The initial seed vector has at least one nonzero component in the direction of the principal eigenvector.

In the list of facts resulting from Perron's Theorem, fact 2 satisfies the first condition of the power method and fact 4 along with an initial vector with at least one positive entry satisfies the second condition. Therefore, the power method will converge to a solution for the principal eigenvector X . The computation using the power method with the terms optimized to take advantage of the sparsity of $D\Delta$ follows.

$$\text{Step 1: } X'_t = D\Delta X_{t-1} + \gamma P; \quad \text{Step 2: } X_t = \frac{1}{\|X'_t\|_1} X'_t \quad (3.3)$$

Figure 3.3b displays the result of applying the above algorithm to the graph in Figure 3.3a. For this example, we use a single constant damping factor of $\Delta = 0.85I$ and P is such that only the goal vertex p_1 has a nonzero personalization value.

AssetRank gives⁴ the expected relative importance for the four vulnerabilities: vul_1 and vul_2 are twice as important as vul_3 and vul_4 since patching one of vul_1 or vul_2 has an equivalent effect in denying the goal p_1 as patching both vul_3 and vul_4 .

3.3.3 Vertex-specific damping

In the case of PageRank applied to web pages, the system-wide damping factor δ gives the probability that surfers will stop surfing [10]. They could stop surfing for any number of reasons including having found the desired information or encountering a poor quality web page. The reality is that not all web pages have an equal likelihood to be the end point of a user's surfing. On some web pages almost all of the surfers will continue surfing (for example, search results) while on other pages, almost all of the surfers will stop surfing (for example, a local weather page).

An analogous situation exists for attack graphs. An “attack planner” will more likely stop traversing the attack graph if the vertex represents a privilege that can be easily obtained “out-of-band”. For example, attackers requiring the ability to execute code on a user desktop could use out-of-band methods, such as social engineering, to gain code execution privileges at the level of the user's account, rather than purely technical exploits.⁵ The damping factor accounts for situations where vulnerabilities and other network facts are less likely to be depended upon by attackers, and thus their rank is reduced. For each vertex, all out-of-band at-

⁴All of the experiments in this chapter required a computation time of less than one second on a typical desktop PC and converged in 78 iterations or less. The complexity of the power method depends upon the complexity of matrix multiplication and the number of iterations required. The complexity of naive matrix multiplication is $O(n^3)$. Speed improvements for PageRank computation can also speed up AssetRank computation as long as they do not require the principal eigenvalue to be 1.

⁵The attack graphs we use in this thesis include only technical exploits.

tacks are considered together as a group since considering them separately would have no effect on the ranks of the graph vertices.

In general, the damping factor measures the likelihood that an attack planner will continue traversing the graph. We improve the accuracy of the ranks by not assuming that the planners are equally likely to stop traversing the graph regardless of the vertex they are visiting. Rather than using a single damping factor, we introduce vertex-specific damping factors δ_v and assemble them into the diagonal damping matrix $\Delta = \text{diag}(\delta_1, \delta_2, \dots, \delta_{|V|})$. While the algorithm provides a parameter for damping factors, further research is required to accurately set the parameter. For example, research into the effects of user security training could provide damping factors that model the likelihood of an attacker successfully mounting a social engineering attack against users possessing varying degrees of security competence.

3.3.4 Personalization vector

Weighted dependency relations model attacker preferences and capabilities, and damping factors model out-of-band attacks but these parameters are insufficient in determining a vertex's rank. Network defenders place a higher priority on defending critical servers than noncritical PCs. We use vertex weights as a *personalization value* to represent a vertex's inherent value to network defenders. Network defenders identify the attack assets they desire to deny the attacker by assigning them a personalization value that reflects their importance to the defender's operations. The remaining attack assets are assigned a personalization value of zero which causes the computed AssetRank values to reflect their importance only in so far as they are likely to be used by an attacker to obtain the attack assets that

have been identified as critical to defend.

3.4 Parameter assignment

Attack graph dependencies and attack asset attribute information (such as CVSS metrics obtained from the NVD database) supply the three key components D , Δ , and P of the AssetRank matrix $A = D\Delta + \gamma Pe^T$. In this section we explain how to obtain and set these values. In Section 5.1 we will demonstrate their effect on the asset ranks. The parameter γ sets the influence of the personalization vector which has the effect of opting to favour attack assets closer to the goal versus favouring attack assets closer to the attacker.

3.4.1 Dependency matrix (D)

To model attacker preferences, we assign a *success likelihood* $s(v)$ to every vertex. The success likelihood has a slightly different meaning for the three types of vertices: AND, OR, and SINK.

The SINK vertices represent the ground facts that MulVAL uses when deriving attack paths. The ground facts include the existence of vulnerable software, network routes and the services running on each machine. Every ground fact is assigned a success likelihood. To simplify the demonstration we assign the success likelihood 1 to all nonvulnerability SINK vertices. That is, we assume that if a service exists, it is always up, and that network paths are stable.⁶

⁶Users could assume mobile devices are present intermittently and hence assign a success likelihood to network routes for mobile devices that represent the likelihood that the device will be connected to the network.

CVSS is a standard for specifying vulnerability attributes. Two attributes that are particularly useful in prioritizing attack assets are the base metric of Access Complexity (AC) and the temporal metric of Exploitability (E). For the AC metric, vulnerabilities are assigned a value of high, medium, or low, to indicate the existence of specialized access conditions such as a race condition or configuration setting. When considering the E metric, vulnerabilities are assigned a value of unproven, proof-of-concept, functional, or high, to indicate the current state of exploit maturity. If one attack path in the attack graph depends upon an unproven vulnerability and another attack path depends upon a vulnerability with functional exploit code, the attack assets in the latter attack path (all vulnerabilities and network routes) are more likely to be involved in an attack and so they are more valuable to attackers. Consequently, they also deserve a higher degree of attention by network defenders. While rigorous success likelihood metrics are unavailable, in our experiments we assign a success likelihood $s(v)$ to each vulnerability vertex v according to Table 3.1. The success likelihood indicates the probability that an attacker will successfully exploit the vulnerability.

CVSS exploitability metric	Success likelihood $s(v)$
Unproven	1%
Proof-Of-Concept	40%
Functional	80%
High	99%

Table 3.1: CVSS exploitability metrics and success likelihoods

MuVAL attack graphs also contain rule vertices. These are AND vertices that specify how a privilege may be obtained. The parameter $s(v)$ for AND vertices models the preference of attackers for different attack strategies. For example, two

of the rules describe how network access may be obtained. In the first case, direct network access to a host is obtained if an attacker has a machine and a network route exists from that machine to the intended host. In the second case, multi-hop network access to a host is obtained if an attacker can execute code of his choosing on a victim machine and a network route exists from that machine to the intended host. Since an attack is complicated by multihop access, we assume that the attacker prefers direct routes so we assign a preference score of 1.0 to the direct route and 0.5 to the indirect route. In a similar manner, other rules may be assigned a preference score indicating attackers' preferences. These rule preferences would be set by experts to model different types of attackers (for example, script kiddies or black-hat criminals). Each model would produce different ranks through repeated applications of the AssetRank algorithm.

Finally, MulVAL attack graphs contain derived attack assets. These are OR vertices in an attack graph and they represent choices that an attacker has in order to obtain the attack asset. For example, MulVAL-generated attack graphs include `execCode(machine, account)` vertices stating that an attacker could obtain the ability to execute arbitrary code on `machine` at the privilege of `account`. However, the `execCode` attack asset might be obtained through a choice of multiple routes in the attack graph. These multiple routes are represented by multiple outgoing arcs from the `execCode` vertex, an OR vertex. Not all of these routes are equally difficult to obtain and we make the assumption that attackers prefer easier methods of obtaining the derived attack asset. For example, attackers would favour routes that may be exploited with reliable tools.⁷

⁷Users of our system can make their own assumptions about attacker preferences and could, for example, assume that attackers will favour routes that utilize theoretical vulnerabilities that do not have published exploit code.

Attack paths will contain several ground facts (SINK vertices), rules (AND vertices), and derived attack assets (OR vertices). Weights of the out-going arcs are computed by percolating the success likelihoods throughout the graph by setting $g(u, v) = m(v)$ where

$$m(v) = \begin{cases} s(v), & \text{if } h(v) = \text{SINK} \\ s(v) \prod_{w \in N^+(v)} m(w), & \text{if } h(v) = \text{AND} \\ \max_{w \in N^+(v)} m(w), & \text{if } h(v) = \text{OR} \end{cases} \quad (3.4)$$

In words, the arc weight from vertex u to v is the success likelihood of v if v is a SINK vertex, the attacker's preference for the attack type multiplied by the product of all of the paths required for v if v is an AND vertex, and the easiest path from v if v is an OR vertex. Finally, the arc weights are normalized according to Equation (3.2).

3.4.2 Damping matrix (Δ)

In Section 3.3.3 we introduced vertex-specific damping factors. This extension allows the modeling of out-of-band attacks for derived attack assets (OR vertices). For example, the ability to execute code on a victim's machine can be gained by obtaining the victim's login credentials through social engineering — a nontechnical attack that is not captured in the attack graph. If attackers gain the attack asset v by means outside the graph, they will not require the dependencies of v captured in the attack graph so those dependencies are less valuable to the attacker and so deserve less attention from network defenders.

For MulVAL attack graphs, specifying a damping factor is only sensible for OR vertices (derived attack assets). The damping factor has no effect on SINK ver-

tices because they have no out-going arcs. Also, AND vertices are fundamentally required in the attack graph and cannot be obtained out-of-band so the damping factor for AND vertices is set to 1 (no damping).

The success likelihood of obtaining a derived asset out-of-band for an OR vertex v is denoted $s(v)$. An example of an out-of-band attack is an attacker obtaining a user's login credentials through social engineering. The success likelihood depends upon the level of awareness and training of the user. A network defender can specify the success likelihood based upon the type of user account. For example, root users could be assigned a low likelihood score such as 20% while standard users could be assigned a score of 80%. Security experts will be relied upon to provide metrics for out-of-band attacks.

The degree to which attackers will use out-of-band attacks depends upon both the projected success of the out-of-band attack and the difficulty of obtaining the attack asset by using the means specified in the attack graph. If the attack asset may be obtained with certainty using the attack graph then the attacker will use those means. Also, if out-of-band attacks are impossible or are certain to fail, the attacker will not exit the graph to attempt the out-of-band means but will use the means in the attack graph to obtain the privilege. The following equation captures these requirements. For an OR vertex v with an out-of-band success likelihood $s(v)$, the damping factor δ_v is given by

$$\delta_v = (1 - s(v)) + s(v)m(v) . \quad (3.5)$$

The damping matrix is a diagonal matrix constructed from the vertex-specific damping factors by setting $\Delta = \text{diag}(\delta_1, \delta_2, \dots, \delta_{|V|})$.

3.4.3 Personalization vector (P)

The personalization vector P represents the network defender's desire to deny an attack asset to attackers. If a defender is only interested in denying a single goal vertex g then its personalization value $f(g)$ is set to 1 and all other vertices are set to 0.⁸ If the defender desires to deny several vertices (for example, the `execCode` privilege on all servers) then the values will be set for the vertices in a manner that represents the defenders (conversely, the attackers) interest in those vertices. It is expected that the defender will set the personalization values based upon the organization's operational priorities.

3.5 Stochastic interpretation of AssetRank

In this section we describe a stochastic interpretation for the numeric value computed by AssetRank on dependency attack graphs. Stochastic interpretation has been used to give the original PageRank a semantic meaning in a random walk model [10, 67]. A random walker surfs the web graph in the following manner. At each time interval, with probability δ it will follow one of the links in the current page with equal probability; with probability $1 - \delta$ it will “get bored” and jump to one of the pages in the web graph with equal probability. Under this interpretation, the equilibrium point of sequence (3.3) will be the probability a random surfer is on a page. This random-walk model cannot be applied to dependency attack graphs, primarily because it does not handle AND and OR vertices differently.

⁸Technically, the non-goal vertices are set to an arbitrarily small $\epsilon > 0$ and the goal is set to $1 - (|V| - 1)\epsilon$. This ensures that the AssetRank matrix $A = D\Delta + \gamma P e^T$ is positive, a condition that guarantees the existence of a unique positive eigenvector according to Perron's theorem.

In this section we give an interpretation of AssetRank that provides meaningful semantics in the context of dependency attack graphs.

Our interpretation is inspired by the model used by Bianchini *et al.* [10]. The defender has the attack graph and is planning how to attack his own network⁹. He does so by dispatching an army of “attack planning agents” whose task is to learn how to obtain the privileges represented by the vertices. Every agent behaves in the following manner: at each moment an agent considers only one vertex in the attack graph. We use $v_i(t)$ to denote the vertex agent i is contemplating at time t . Let $v = v_i(t)$. If v is a sink vertex, agent i has finished his job and stops working. Otherwise he will, with probability δ_v , plan how to satisfy the requirements for v based on the attack graph; with probability $1 - \delta_v$, he stops traversing the graph and decides to obtain the privilege v through other means not encoded in the attack graph (for example, through backdoors already installed in the system or social engineering). In the latter case, the agent has also finished his planning and stops working.

With probability δ_v , the agent uses the attack graph and follows the out-going arcs from v to satisfy its preconditions. Two cases need to be considered. If v is an OR vertex, the agent will choose one of its out-neighbours w with the following probability.

$$Pr[v_i(t+1) = w \mid v_i(t) = v] = g(v, w) \quad (3.6)$$

If v is an AND vertex, the agent must plan how to satisfy *all* the out-neighbours of v . Thus he must move along all the out-going arcs simultaneously. We model

⁹Building the attack graph requires a great deal of information about the network and so the defender's penetration tester, in possession of the attack graph, represents the most capable attacker in the class given by the attacker model used to produce the attack graph.

this by allowing the agent to replicate itself¹⁰ with each replica moving to one of the out-neighbours independently. More precisely, at step $t+1$ agent i will become $r = |N^+(v)|$ agents i_1, \dots, i_r , each of which is assigned one of the vertices in $N^+(v)$ so that every element in $N^+(v)$ is covered.

The attack planner has an unlimited number of such agents at his disposal. Every time he dispatches an agent to a vertex in the attack graph, the agent will try to find a way to attack the system such that the goal represented by the starting vertex can be achieved. When the agent (and all his clones) finishes the job, an attack plan has been made. Each time he may find a different attack path due to the probabilistic choices he makes along the way. At each time interval, the attack planner will dispatch new agents with probability γ and the new agents will start from one of the graph vertices with the probability distribution specified by the personalization vector P . The number of new agents is γ times the number of active agents currently in the system.

Let the vector $X_t = [X_t^1, \dots, X_t^{|V|}]^T$ where X_t^v is a random variable representing the number of active agents planning an attack for vertex v at time t . Let the vector $E(X_t) = [E(X_t^1), \dots, E(X_t^{|V|})]^T$ where $E(X_t^v)$ is the expected value of the random variable X_t^v . Let $E(X_0) = P$ which corresponds to the attack planner dispatching the first agent according to the probability distribution given by P . The following equation then holds for $t > 0$.

$$E(X_t) = D\Delta E(X_{t-1}) + \gamma \|E(X_{t-1})\|_1 P \quad (3.7)$$

After normalization, this is precisely the sequence specified by (3.3). The normalized value of $E(X_t)$ converges to a unique solution as $t \rightarrow \infty$ and the AssetRank

¹⁰Analogous to the UNIX `fork()` command.

value computed will represent the proportion of active attack planning agents on each vertex in the attack graph.

Under this attack-planning-agents interpretation, a higher AssetRank value for a vertex indicates there will be a larger portion of planning agents discovering how to obtain the asset represented by the vertex. Thus, our AssetRank metric directly implies the importance of the privilege or vulnerability to a potential attacker in reaching the defence goals. The arc weight $g(v, w)$ indicates the desirability of the attack step (v, w) with respect to achieving the capability v , since a higher $g(v, w)$ means a planning agent will be more likely to choose w as v 's enabler. A vertex's personalization value represents the desire of the defender to deny the privilege to attackers. A higher personalization value indicates the vertex is more important to the defender's attack planner and so he is more likely to dispatch a planning agent to determine how to achieve the goal. A lower δ_v indicates an attacker is more likely to gain privileges by out-of-band means and thus will not follow the attack graph. γ indicates the rate at which the attack planner dispatches new agents.

3.6 Summary

In this chapter we proposed the AssetRank algorithm, an eigenvector ranking algorithm for directed F-graphs that can be applied to rank the importance of a vertex in a dependency attack graph. The model gives the ability to reason on heterogeneous directed graphs containing both AND and OR vertices. It also adds the ability to model various types of attackers. We have shown how to incorporate vulnerability attribute information into the arc weights. Similarly, users could compute attack-asset ranks derived from metrics regarding attack noisiness, attack-path

length, or resource utilization. We have also shown how to model the existence of out-of-band attacks into vertex-specific damping weights. We incorporated personalization values to allow network defenders to specify the assets that they most desire to deny attackers and thus obtain a personalized attack-asset ranking based upon their operational priorities.

The numeric value computed by AssetRank is a direct indicator of how important the attack asset represented by a vertex is to a potential attacker. The algorithm is empirically verified in Section 5.1 through numerous experiments conducted on several example networks. The rank metric will be valuable to users of attack graphs by giving a better understanding of the security risks, in fusing publicly-available attack-asset-attribute data, in determining appropriate mitigation measures, and as input to further attack-graph analysis tools.

It has been recognized that the complexity of attack graphs often prevents them from being useful in practice and methodologies have been proposed to better visualize them [37, 62, 63, 87]. The ranks computed by our algorithm could be used in combination with the techniques in those works to help further the visualization process, for example by coloring the visualization based on the computed ranks.

Chapter 4

Partial-cut vertex sets

4.1 Introduction

Modern network environments are extremely complex, as is the job of ensuring their security. Network administrators must defend assets with varying priorities against attacks of diverse complexity and maturity. Additionally, they have only finite resources at their disposal and they face the difficult task of balancing security, usability, and resources. We present a solution that supports network administrators in making best-case security decisions that optimize resource utilization and maximally defend their network, within a specified budget.

Considerable work [4, 39, 41, 64, 65, 77, 81, 82] has been done in recent years to develop automated security analysis tools that compute multihop network attacks based upon network configuration data such as host connectivity, installed software, services offered, and open source vulnerability databases. As mentioned in previous chapters, a computationally practical method describes complex network attacks by the use of *dependency attack graphs*. Dependency attack graphs

illustrate how low-level details of the network configuration can be chained together by an attacker to compromise network assets.

Attack graphs may be generated with the attacker starting at any network node (including internal nodes) and targeting any number of nodes. Since the information describing a successful attack is contained in the graph, a network administrator could, in theory, use an attack graph to decide on an ideal course of action (COA) to improve the security of the network by making both strategic changes, for example by planning a software vendor switch, and tactical changes, for example by severing a particularly vulnerable network connection. However, while it is simple to consider any individual attack step in the graph, analyzing the global graph is overwhelmingly complex. Hence, the network defender is inundated with information and unclear about how best to proceed. Furthermore, zero-day attacks¹, resource limitations, and operational constraints make it very difficult to completely secure a network [55]. Network defenders require a solution that maximally, but not necessarily completely, improves security within real-world constraints.

To be useful in practice, COA recommendations should take into account four factors. The first two are closely related and are tied to operational priorities. First, large organizations execute many operations and those operations have different priorities. Adding complication is the fact that the priorities of operations change over time, as do the priorities within the operation. For example, an airline website selling tickets before the busy Christmas season might need to prioritize the availability and performance of their ticket purchase system in November. Once the high-volume buying time period has ended, the availability and performance of

¹A zero-day attack exploits an undisclosed software vulnerability.

the online check-in system becomes a high priority. The availability of guest email stations at the airport would be a lower priority than both the ticket purchase and online check-in systems. Since some network services are more valuable to the defender than other services, their protection should be prioritized accordingly. Assigning priorities to services can be done with AssetRank, another algorithm or heuristic, or manually.

The second factor follows from the prioritization specified in the first factor and it is that some network configuration changes are more costly than others. We use the concept of a *loss function* [34] to provide a *cost* corresponding to the removal of each SINK vertex. A loss function is a non-negative function mapping an event (a configuration change) to a real number (the cost incurred in making the configuration change). Determining the financial cost, time investment cost, and denial-of-use cost² for network configuration changes is complicated to the point that it is a body of work in its own right, called decision theory. In our experiments we use simple loss costs but our algorithms are flexible enough to allow any linear loss cost as an input. While our work is limited to loss costs that add linearly, a cost could reflect the total for changing all vertices of a particular group. For example, a vertex cost could comprise the total for patching all network nodes with the same vulnerability. Also, the user can iteratively increase or decrease the cost in response to COA recommendations. If a network fact cannot be changed (*e.g.*: vulnerable critical software without a patch available), its cost is considered infinite.

The third factor to consider is that the network administrator has a limited

²Insofar as certain changes will disrupt network function and impair the organization's business activity.

budget to expend on these costs. Except for very simple networks, it is unlikely that all of its security problems can be fixed; the administrator aims to maximally improve security, given the budget available.

Fourth, some vulnerabilities and privileges are more useful to attackers than others due to the ease of exploitation, stealthiness of the attack, capabilities they give attackers in furthering the attack, or other factors, and one ought to preferentially obstruct the attackers' ability to exploit or acquire the most useful privileges. We model the dependence that attackers are likely to have on each attack asset by assigning a rank weight using AssetRank to every vertex in the attack graph. Note that AssetRank can be used twice in this problem set. The first application is on a dependency graph that captures the dependencies between operations and services. The second application is on a dependency graph that captures the dependence between attacks and network configurations (an attack graph). In the context of network attack graphs, the AssetRank value assigned to each vertex represents the relative dependence an attacker places on that vertex in furthering his attack against the network. As such, the defender's objective should be to preferentially eliminate vertices with high AssetRank values. Any rank function whose ranks can be linearly aggregated may be used as an alternative to AssetRank in our partial-cut algorithms that give COA recommendations.

We solve the network security problem discussed above by considering the general problem of reducing connectivity in AND/OR directed graphs. In the general case, our objective is to maximally decrease the connectivity between two disjoint communities in an AND/OR directed graph by deleting vertices. Each deletion has a distinct cost and the sum of all costs must not exceed a budget. For attack graphs, this corresponds with maximally decreasing the connectivity between

the goal vertices and the attacker vertices. AssetRank values reflect the degree to which the attacker depends upon each vertex in reaching the goal set.

In Section 4.3 we introduce the *closure* of a vertex set in AND/OR directed graphs and give a linear-time algorithm for its computation. Section 4.4.1 defines our optimization function for the reduction of graph connectivity and Section 4.4.3 presents an exponential-time brute-force algorithm to compute partial-cut vertex sets. In Section 4.4.4, we introduce *closure-relation graphs*. The graphs give insight into the connectivity-reduction problem and enable us to create a polynomial-time algorithm to compute an approximation to the ideal solution given by the brute-force algorithm. In Section 4.5.4 we show that finding perfect solutions is NP-hard and thus, an approximation to the perfect solution is the best we can do in polynomial-time, unless $P = NP$.

4.2 Generation of scale-free networks

In order to direct the discussion in this thesis, the graph concepts are applied to computer network defence. The ability to generate networks of any size is useful when testing the scalability and applicability of algorithms; in particular, the graph connectivity algorithms in this chapter. Albert and Barabási [2, Table 2] summarize the indegree and outdegree exponents for twenty kinds of networks with a power-law distribution. Researchers have modelled inter-domain networking using undirected graphs because it accurately models routing. Doyle *et al.* [21] show that while Internet router connectivity follows a power-law distribution, it is not accurately modelled by a scale-free model. In our case, our attention is not on the physical infrastructure and whether domains are neighbours, but rather

our concern is whether the network access control lists (ACLs) allow inbound and outbound connections.

The most important factor in attack propagation is whether or not a communication session may be set up. If a network allows outbound session initialization to another network, it does not imply that it allows inbound session initialization. For example, a network likely allows any host to initiate a communication session with Google; however, the converse would rarely be true.

Bollobas *et al.* [11] introduce a model for generating scale-free directed graphs using preferential attachment. Conceptually, preferential attachment is the notion that vertices with many in-neighbours are more likely to receive new connections than vertices with few in-neighbours. Likewise, vertices with many out-neighbours are more likely to make a new connection than vertices with few out-neighbours. For the WWW, we see this reflects reality. If a page has many outbound links (for example, a directory page), it is likely to add links in the future. Similarly, if a page has many incoming links, it is likely that additional pages will link to the page in the future. Network connectivity follows the same pattern. A server that already allows many inbound connections (for example, a DNS server) is likely to allow connections from newly added hosts. Similarly, a server that is allowed to access many other hosts (for example, a patch deployment server) will likely be allowed to access new hosts as well.

In the absence of published data giving verified power-law exponents on network connectivity, we use the values that Bollobas *et al.* [11] suggest for web graphs. Empirical evidence shows that enterprise network subnets are more likely to initiate outbound connections than to receive them, and this is also the behaviour of web graphs. The power law exponents for web graphs are $\gamma_{out} = 2.7, \gamma_{in} = 2.1$.

Our purpose in generating network connectivity models is to provide a network for which we may generate AND/OR directed attack graphs for scalability testing and algorithm comparison. Should the scale-free directed graphs have shortcomings in modelling network dependencies, the algorithm comparison of scalability will still be valid.

Los Alamos National Lab provides an open source Python module for graph manipulation and analysis called NetworkX [31]. The package implements a scale-free graph generator whose details are published in the Bollobas *et al.* paper [11]. The scale-free graph generator in NetworkX produces graphs with self-loops; however, this feature does not meet our requirement since we assume *all* network hosts have unrestricted self-communication. Therefore, we modified the graph generation code to avoid self-loops.

Our target is to generate and analyze AND/OR directed graphs that give the attack graph for a network. This involves extending the NetworkX directed graph class to handle the semantics of AND/OR directed graphs plus the following four steps:

1. Generate a computer network model;
2. Select an attacker starting location;
3. Select the network assets to defend; and
4. Generate the attack graph.

First, a directed scale-free graph is generated. Each vertex of the graph can be thought to represent a host class that is a collection of abstracted hosts which all

share a single baseline of software and network connectivity. Each host class contains between one and three vulnerable services that are listening on a remotely accessible port and will respond to hosts that have the ability to establish a connection. In enterprise networks, a host class could represent all the hosts in a subnet that share a common baseline, or individual servers (which do not share a common baseline). A directed edge in the network graph represents that the source host has network access to the target host on any port.

Before generating an attack graph, we first randomly select the attacker's starting location from a discrete uniform distribution. A random starting location is chosen for the attacker because attackers could start at any machine via any number of vectors including social engineering, shared malicious software, or a malicious insider. Due to the artifacts of preferential attachment, it is most likely the case that the attacker location will correspond to a workstation that does not accept incoming connections but occasionally it will correspond with a popular website or server.

Next, a user-specified number (say k) of high priority hosts is chosen via one of two methods we have programmed. In the first method, the PageRank algorithm is used to compute a rank value for all hosts in the network. Recall that the semantics of ranks computed by the PageRank algorithm are that each dependency is disjunctive (represents alternatives). The k hosts to defend are chosen to be the k hosts with the highest PageRank. Our assumption is that these hosts correspond with the servers in the network that are essential to its functioning; this method takes into account cascading dependencies. In the second method, the k hosts to defend are heuristically chosen to be the k hosts with the highest number of incoming connections. Our assumption is that these hosts correspond with the

Number of incoming connections	Host numbers
0	26, 27, 25, 22, 23, 20, 21, 29, 7, 6, 5, 4, 3, 19, 18, 17, 16, 15, 14, 13, 12, 11
1	24, 28, 9
2	1
3	10
4	8
7	0
20	2

Table 4.1: Distribution of incoming subnet connections

Number of outgoing connections	Host numbers
0	24, 28, 8
1	26, 27, 25, 22, 23, 21, 29, 7, 4, 2, 9, 19, 18, 15, 14, 12, 10
2	20, 6, 3, 1, 0, 17, 16, 13, 11
4	5

Table 4.2: Distribution of outgoing subnet connections

servers in the network that contain frequently required operational services and so are essential to its functioning.

Since the attacker location is chosen randomly, it can happen that the attacker is already located on one of the high priority hosts. In that case, the other k highest priority hosts are selected as goals to defend. The attack graph is generated using the MulVAL framework.

Figure 4.1 shows a sample randomly generated 30-host-class network. Tables 4.1 and 4.2 give the distribution of incoming and outgoing node connections for the network.

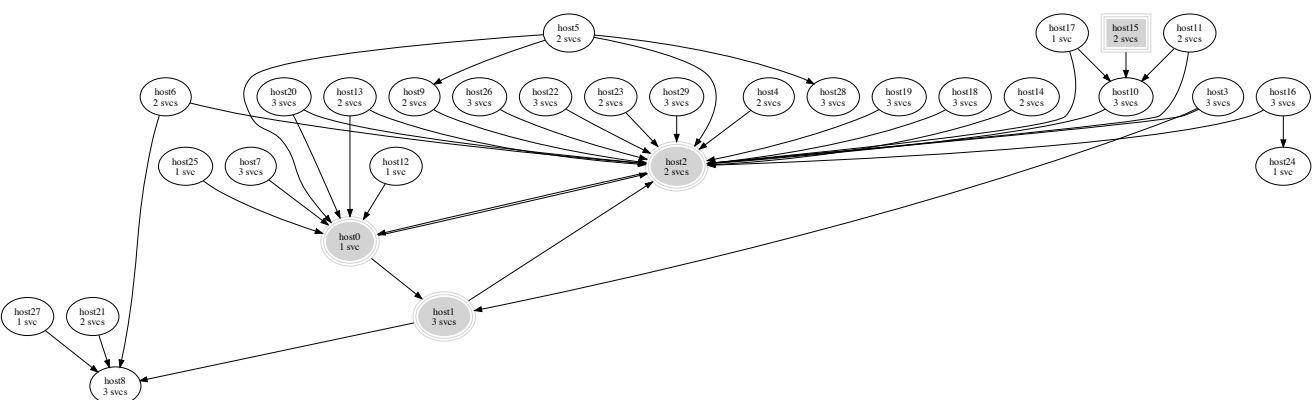


Figure 4.1: Randomly generated scale-free computer network with 30 hosts. The attacker starts at host 15 and the target hosts are 0, 1, and 2.

4.3 Closure of AND/OR directed graph vertex set

4.3.1 Discussion and definition

Given a dually weighted³ AND/OR directed graph, we consider how to generate optimal partial-cut vertex sets under a budgetary constraint. The approach we take is to consider the partial disruption resulting from removing SINK vertices in contrast to considering the negation of logic clauses, which corresponds with complete graph cuts. When a SINK vertex is made invalid, the portion of the graph that depended upon it (representing derived privileges) will become invalid, meaning that it is of no use to the attacker in reaching the goal vertices. Summing the AssetRanks of all of the invalid vertices gives a metric indicating the effect on the attacker of removing the SINK vertex (*i.e.*, the proportion of rank that becomes inaccessible to the attacker). However, the determination of invalid vertices is a cascading problem.

Initially, all vertices in the attack graph are on a path from a goal vertex to an attacker vertex. If we delete the vertices that are invalid (and the incident edges), vertices remaining in the graph could become unreachable from the goal vertex, or unable to reach the attacker vertex. Each vertex that is not in a path from the goal set to the attacker does not aid an attacker to reach the goal. A disconnected subgraph is an example of both unreachability cases.

As an example, consider the attack graph shown in Figure 4.2a. If vertex i is deleted, then the compound arc⁴ incident to it is also removed, resulting in the graph shown in Figure 4.2b. Vertex j is still present but it is not useful to the at-

³Rank weights for all vertices and cost weights for SINK vertices.

⁴Note that every AND vertex has a *single* outgoing compound arc. Invalidating any one of an AND vertex's out-neighbours deletes the arc to every out-neighbour.

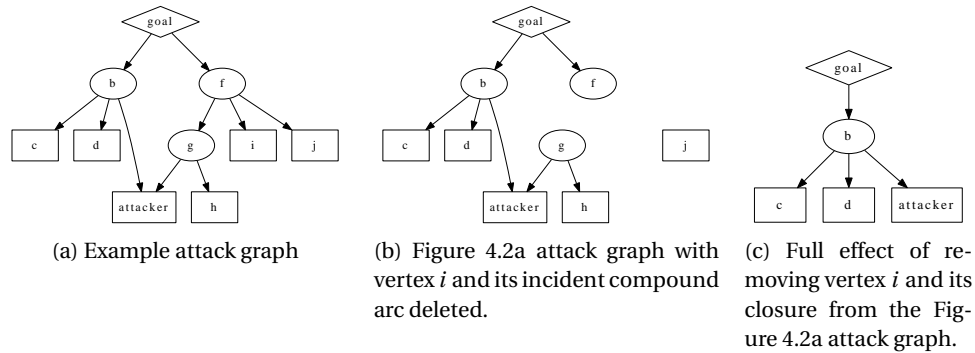


Figure 4.2: Determining the effects of vertex removal. AND vertices are denoted by ellipses and correspond to attack methods; OR vertices are denoted by diamonds and correspond to new capabilities the attacker can derive; SINK vertices are denoted by boxes and correspond to network configuration facts.

tacker because, being disconnected, it is no longer reachable. We see that g can reach the attacker but is not reachable by the goal, and so it too is no longer useful. Removing g isolates h so it should also be removed. Vertex f is reachable from the goal but cannot reach the attacker so it is not useful. Figure 4.2c shows the final effect of deleting vertex i . In this example, the partial cut-set is $\{i\}$, and corresponds with a COA recommendation. We term the set of vertices that are deleted as a result of deleting the partial cut-set the *closure* of the set. In this example the closure of $\{i\}$, denoted $\overline{\{i\}}$, is $\{f, g, h, i, j\}$. We summarize the preceding discussion by formally defining a vertex set closure in F-graphs.

Definition 4.1 (Vertex Set Closure). Let S (source) and T (target) be disjoint subsets of V in an F-graph $G = (V, A)$, and assume that G is equal to the graph induced⁵ by S and T on G . Then the *vertex set closure* of C , a set of vertices to delete (along with the incident arcs) from G , is defined to be $\overline{C} = V - V'$ where $G' = (V', A')$

⁵Recall Definition 1.1

is the graph induced by S and T on $G - C$.

4.3.2 Vertex set closure algorithm

Our algorithm to compute a vertex set closure is used by both the polynomial and exponential algorithms; the full details are presented in Algorithm 4.1 (VertexSet-Closure). Briefly, the algorithm computes the closure of a set of vertices by:

Line 1 Remove the partial cut-set vertices from the graph.

Line 2 Compute the AND and OR vertices unable to reach the set of attacker starting locations in the directed graph. This is efficiently computed by finding the vertices reachable from the attacker set in the graph transpose.⁶

Line 3 Compute the vertices unreachable from the goal set in the directed graph.

Computing the vertex set closure is very efficient (see Section 4.5). Approaching the problem from the goal-informed but COA-centric point-of-view (“What are the network effects of removing this privilege?”) instead of a goal-centric approach (“What combination of privilege removals is required to falsify the goal?”) enables a polynomial-time algorithm⁷ and an efficient way to measure partial security improvements.

⁶The graph transpose is the graph with all arcs reversed, and is denoted G^T for a graph G .

⁷Polynomial in terms of the number of vertices and arcs.

Algorithm 4.1 VertexSetClosure(G, C, S, T) — Closure of a vertex set

Input: AND/OR directed graph G , set of vertices C whose closure is to be computed, source vertex set S that vertices must be reachable from (in an attack graph, these are the goal vertices), target vertex set T that vertices must be able to reach (in an attack graph, these are the attacker starting locations).

Output: Graph G' , the minimal graph induced by $G - C$ with respect to S and T , and the closure \overline{C} of C .

- 1: $G' \leftarrow G - C, \overline{C} \leftarrow C$
 - 2: Find the AND and OR vertices in G' that cannot reach T , remove them from G' , and add them to \overline{C}
 - 3: Find the vertices in G' that cannot be reached from S , remove them from G' , and add them to \overline{C}
 - 4: **return** G', \overline{C}
-

4.4 Partial-cut vertex sets in AND/OR directed graphs

4.4.1 Discussion and definition

It is important to note that the direct removal of a set of COA vertices from a graph can affect legitimate network users because their removal signifies changes to the network configuration; whereas the indirect removal of the remaining vertices in the closure of the COA set only affects attackers. In the previous example, the removal of SINK vertex i can affect both attackers and users, whereas the SINK vertices j and h are no longer useful to the attacker and so are removed from the attack graph but their removal does not cause configuration changes in the network.

Our aim is to compute the partial-cut vertex set with the least cost, within a budget, that removes the maximum rank from the graph. Each SINK vertex has an associated cost given by a loss function, so the cost for the partial cut-set is defined as the sum of the costs of its member vertices. Likewise, each vertex in the graph

has a rank value, and we define the rank of the partial cut-set closure⁸ as the sum of the ranks of its members. Formally, our optimization problem is to find a subset C of the SINK vertices such that:

$$\max_C \left(\sum_{c \in \bar{C}} \text{rank}(c) \right) \quad (4.1)$$

under the constraint

$$\sum_{c \in C} \text{cost}(c) \leq \text{Budget} \quad (4.2)$$

and additionally, that C is the least cost such set.

In vector notation, if $|V| = n$, then let $C \in \{0, 1\}^n$ be an indicator vector of vertices (in particular, SINK vertices) to remove from the graph, R a vector containing the ranks of the vertices in V , and K a vector containing the vertex removal costs. Since, in attack graphs, we limit ourselves to removing SINK vertices⁹, the cost of every non SINK vertex is infinity. We want to find C such that:

$$Q = \max_C (\bar{C} \cdot R) \quad (4.3)$$

under the constraints:

$$ROI = \max_{\{C | \bar{C} \cdot R = Q\}} (\bar{C} \cdot R / C \cdot K) \quad (4.4)$$

$$C \cdot K \leq \text{Budget} \quad (4.5)$$

⁸Note that we are interested in the cost of the partial cut-set but rank of its closure.

⁹Recall that SINK vertices correspond with network facts that can be changed by the network defender while AND and OR vertices correspond with derived privileges the attacker gains by exploiting network facts.

4.4.2 Alternative rank and loss costs

Our approach is flexible due to the use of rank and cost weights. Rank weights may reflect whatever the defender wants to deny the attacker, and cost weights may reflect whatever the defender wishes to change. For example, in place of the expressive rank and loss function cost values we presented, the defender could assign a uniform rank value to the network services and set all other vertex ranks to zero. If the defender assigns a cost of 1 to each vulnerability vertex and set the remaining SINK vertices to an infinite cost, the computed COA would then maximally deny the attacker access to network services by patching as few vulnerabilities as possible. The flexibility and efficiency of our approach should be very useful in practice.

4.4.3 Brute-force partial graph cut algorithm

In Section 4.5.4 we show that, in general, solving Equation 4.3 is NP-hard by reduction from the supermodular knapsack problem. Algorithm 4.2 computes the optimal solution by brute force. It considers all combinations of vertices that are within budget. That is, all sets with one vertex are considered, as are all combinations of two vertices, and so on until all combinations (within budget) are exhausted. If the number of in-budget SINK vertices in an attack graph is n , there are 2^n combinations to consider. Thus, Algorithm 4.2 has exponential complexity (see Section 4.5) and it is not practical for finding complete solutions in large graphs.

A formal proof of the total correctness of Algorithm 4.2 would require that both the specification and algorithm are given in a formal language. However, we can

sketch a proof of total correctness that may satisfy the reader.

Proposition 4.1. *Algorithm 4.2 is totally correct with respect to the specification given by Equations 4.3, 4.4, and 4.5.*

Proof sketch. A proof of total correctness requires both a proof of correctness according to the specification and that the algorithm always terminates.

Correctness

Line 2 of Algorithm 4.2 generates all possible candidates for C of Equation 4.3, and the budget condition given in Equation 4.5 is also ensured with line 2. Line 3 of the algorithm performs an exhaustive loop over the candidates generated in line 2 while the summation in line 5 computes $\overline{C} \cdot R$ of Equation 4.3 and line 6 enforces the max function; thus the only solutions kept in the end are those with a maximum rank-sum (i.e., $\{C \mid \overline{C} \cdot R = Q\}$). The ROI (Equation 4.4) is maximized when the cost $C \cdot K$ is minimized. Since the retention of only the minimum cost vertex sets is performed in line 11, the ROI is maximized. Therefore, the returned CandidateVertices is consistent with C , and RankEliminated is consistent with Q in Equation 4.3, and the return values satisfy the constraints in Equations 4.4 and 4.5.

Termination

Since the size of the list generated in line 2 of Algorithm 4.2 is finite and the loop in line 3 terminates after iterating over the finite collection, the algorithm always terminates. □

Algorithm 4.2 BruteForcePCVS(G, S, T, Budget) — Partial-cut vertex set

Input: AND/OR directed graph $G = (V, A)$, goal vertex set S , attacker vertex set T , Budget.

Output: PCVSet, RankEliminated

- 1: CandidateVertices $\leftarrow \emptyset$, RankEliminated $\leftarrow -\infty$
- 2: CandidateVertices $\leftarrow [C \in \text{PowerSet}(V), \text{cost}(C) \leq \text{Budget}]$
- 3: **for** $(PCVS_i, \text{Cost}_i) \in \text{CandidateVertices}$ **do**
- 4: $(G_i, \overline{PCVS}_i) \leftarrow \text{VertexSetClosure}(G, PCVS_i, S, T)$
- 5: $PCVSRank_i \leftarrow \sum_{v \in \overline{PCVS}_i} \text{Rank}(v)$
- 6: **if** $PCVSRank_i > \text{RankEliminated}$ **then**
- 7: RankEliminated $\leftarrow PCVSRank_i$
- 8: CandidateVertices $\leftarrow \{PCVS_i\}$ {Replace previous solutions}
- 9: **else if** $PCVSRank_i = \text{RankEliminated}$ **then**
- 10: CandidateVertices $\leftarrow \text{CandidateVertices} \cup \{PCVS_i\}$ {Set of sets}
- 11: Keep only least-expensive solutions in CandidateVertices
- 12: **return** CandidateVertices, RankEliminated

4.4.4 Closure-relation graphs

We improve on the approach in the brute-force algorithm by recognizing there is an underlying graph, which we call the closure-relation graph. It is the graph whose vertices are each a vertex set closure, and the arcs represent a vertex deletion that links closures. Consider the closure-relation multi-digraph in Figure 4.3, which is computed from the attack graph given in Figure 4.2a. The arcs of the closure-relation graph indicate the vertex deletions (COA steps) that result in new unique subgraphs of the attack graph. The attack graph subgraphs are formed by deleting the closure corresponding with that vertex of the closure-relation graph from the original attack graph. Figure 4.4 is an alternate digraph representation where the vertex labels show the remaining attack graph and vertices are connected by a single arc that is labelled by the choice of equivalent attack graph ver-

tices whose removal transitions between the adjacent vertices.

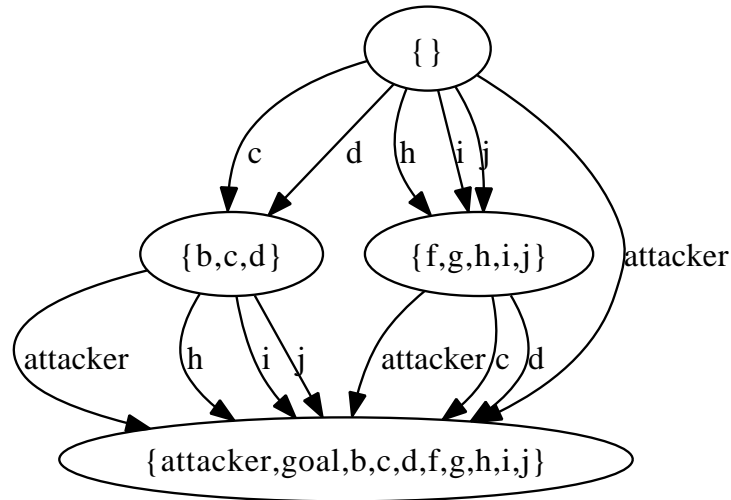


Figure 4.3: Closure-relation graph for the attack graph in Figure 4.2a. Each vertex is labelled by the closure resulting from removing the attack graph vertex labelled on the incoming paths. Each arc corresponds to the deletion of a single attack graph vertex.

Recall that a COA is a set of mitigations that are implemented all together. By using the closure-relation graph, we build up COAs by considering a single vertex at a time. This greatly simplifies the task of building multi-vertex COAs by transforming it to a problem of traversing paths in the COA graph. For example, since a COA is a set, order is not important and so from the graph in Figure 4.3 we can find the closure of the combination $\{c, h\}$ by either first traversing the arc c , then h , or by first traversing h , then c . Thus, the closure-relation graph nicely reflects the commutative property of SINK removal.

The closure-relation graph also clearly shows the removals that are equivalent. Looking at the out-arcs of the root null-set vertex, we see that removing h , i , and j all result in the same vertex deletion set closure.

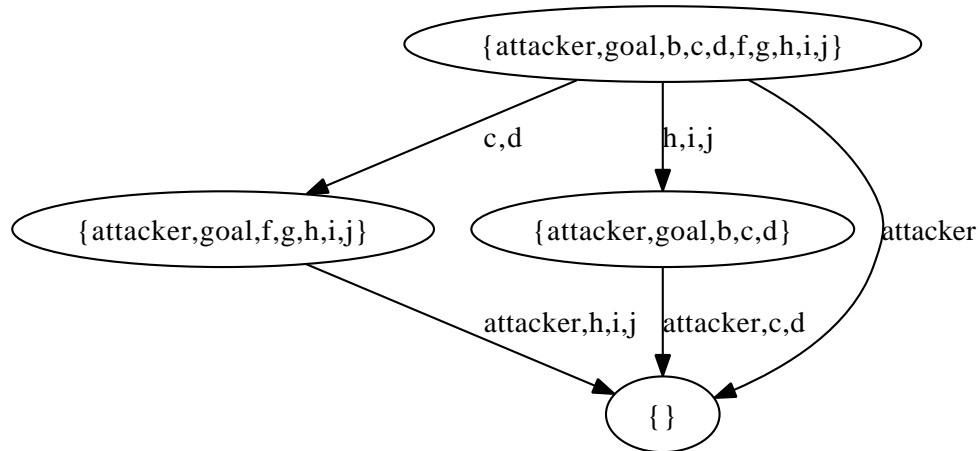


Figure 4.4: Alternate representation of the closure-relation graph in Figure 4.3. The vertex labels show the remaining attack graph and vertices are connected by a single arc that is labelled by the choice of equivalent attack graph vertices whose removal transitions between the adjacent closure-relation-graph vertices.

Assuming the closure-relation graph is built and each edge weight is the deletion cost of the vertex represented by the edge, we can use variants of Dijkstra's algorithm¹⁰ to find the following information. In each case, the source is the null closure vertex (root vertex).

1. **Least cost full solution:** Given by the shortest weighted path from the null closure vertex to the supremum closure vertex containing all vertices in the attack graph.
2. **Best solution within a budget:** Given by computing the shortest path tree using the budget as a stopping condition. For each vertex in the closure-relation graph, the sum of the rank weights of the closure vertices is com-

¹⁰Edsger Dijkstra invented the algorithm bearing his name in 1959. Given a graph with non-negative edge weights, the algorithm efficiently computes the least-expensive path from a single source to all other vertices.

puted. The leaf (or leaves) with the largest rank-sum gives the optimal closure and the path from the null closure vertex gives the COA.

The closure-relation graph gives insight into the problem of computing COAs but building the full graph can be an even more complex way of computing the optimal solution than the brute force algorithm developed previously. On the one hand, the previous brute-force algorithm would have computed every vertex deletion combination (for example, referencing the just-mentioned arcs, the brute-force algorithm would have tried the combinations $\{h\}$, $\{i\}$, $\{j\}$, $\{h, i\}$, $\{h, j\}$, $\{i, j\}$, and $\{h, j, j\}$ but from the closure-relation graph we know these combinations are all equivalent since the attack graph vertices have the same source and target in the closure-relation graph. Thus, if there are k arcs with the same source and target, the closure-relation-graph algorithm computes k closures while the brute-force approach computes the closure of 2^k combinations. The computational impact depends upon the size of k for the graph under consideration. If k is small, the computational time difference would be negligible but as k grows the closure-relation-graph approach provides a significant speedup when many COAs are equivalent.

On the other hand, there can be many paths to a closure vertex in the closure-relation graph. For example, if a vertex deletion set is comprised of $C = \{1, 2, 3\}$ and none of the vertex deletions 1, 2, or 3 are equivalent, the paths in the closure-relation graph to reach the closure of C are: $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, and $[3, 2, 1]$. In general, if there are j elements in a vertex deletion set, there are $j!$ (j factorial) ways to navigate the closure-relation graph to learn the closure of the set. The savings obtained by reducing the number of equivalent vertices considered is

more than offset by the additional cost of including the permutations of long paths since factorials grow much faster than exponentials.

Therefore, we do not build the full closure-relation graph to compute solutions but instead use the graph as a theoretical help to better understand the problem and to derive algorithms that explore the graph in an efficient manner.

4.4.5 Best-first search partial-cut algorithm

We improve on the brute-force algorithm by considering how to build a subgraph of the closure-relation graph by exploring a path in an expedient manner. The “best-first search” (BFS) is a heuristic that is used to expand the graph by selecting the most profitable vertex according to an evaluation function. The evaluation function we use is Equation 4.4 but we change the scope of the max function to range over all attack graph vertices in a subgraph of the original attack graph. That is, C is an indicator vector with a single non-zero element.

$$f(C) = \bar{C} \cdot R / C \cdot K \quad (4.6)$$

This evaluation function gives a return on investment (ROI) by computing the amount of rank that would be removed from the graph G with the investment of removing a particular attack graph vertex v . It is evaluated for each SINK vertex v whose cost is within the budget. We compute the rank of the closure \bar{C} in the context of the closure-relation graph G and divide by the cost of the attack graph vertex.

The vertex with the highest ROI is removed from the graph (along with its closure) and the cost is deducted from the available budget. The process is repeated with the resulting subgraph and terminates when the budget is exhausted,

or no further improvements are possible within the unspent budget. After all vertex deletions from a selected source are computed, the least cost to reach every closure-relation-graph vertex is computed by using the single-source Dijkstra algorithm. The path costs are recomputed every time after the vertex deletions from each source vertex are processed. This full recomputation is performed because the cost to reach the graph vertices might have decreased through paths revealed in the exploration. The ROI of each closure-relation-graph vertex that is an out-neighbour of the currently selected source vertex is computed and the highest ROI path is selected. The target vertex along that path becomes the new source vertex and the process continues until it happens that the highest ROI vertex does not have any out-arcs.

The situation where the highest ROI vertex has no out-arcs can occur in two cases. Case 1 is when the highest ROI vertex closure is the entire attack graph. In this case, the return value for the algorithm is the least cost path from the null closure vertex to the supremum vertex. Constructing the least cost path is performed with Dijkstra's algorithm, which is modified to account for the fact that the closure-relation graph is a multi-digraph and so between any two vertices many arcs of equally low cost can exist.

The second case is where no out-arcs exist within budget. In this case, a graph search is performed for the closure-relation-graph vertex with the highest rank-sum; that vertex is then selected as the new source vertex. If the highest rank vertex is also the highest ROI vertex, then the return value is the least cost path from the null closure vertex to the target vertex. Otherwise, graph exploration continues as before. If both the highest ROI and highest rank vertices do not have out-neighbours within budget, the return value is the least cost path from the null clo-

sure vertex to the highest ranked vertex. By computing the least cost to each vertex after each round of exploration, suboptimal vertex deletion sets with higher than necessary cost that were computed by the original BFS algorithm are corrected.

In summary, the BFS algorithm discussed here uses ROI as an evaluation function, and then falls back to the highest rank-sum vertex when it encounters a dead-end in the graph due to reaching the budget. Our best-first search partial-cut algorithm is presented in Algorithm 4.3.

Algorithm 4.3 BFS-PCVS($AG, S, T, Budget$) — Best-first search partial-cut vertex set

Input: AND/OR directed graph $AG = (V_{AG}, A_{AG})$, goal vertex set S , attacker vertex set T , Budget.

Output: PCVS, TotalCost, RankEliminated.

```

1:  $CRG \leftarrow (V_{CRG}, A_{CRG}), V_{CRG} \leftarrow \{\emptyset\}, A_{CRG} \leftarrow \emptyset$  {Initialize closure-relation graph}
2:  $Process \leftarrow [\emptyset], Seen \leftarrow \emptyset, Spent \leftarrow \emptyset$ 
3: while Process is not empty do
4:    $vSourceClosure \leftarrow Process.pop()$ 
5:    $Seen \leftarrow Seen \cup vSourceClosure$ 
6:    $Spent \leftarrow$  cost of shortest path to  $vSourceClosure$  (stored)
7:   for  $v \in \{v \mid v \in V_{AG} - vSourceClosure; Cost(v) \leq Budget - Spent\}$  do
8:      $\overline{C} \leftarrow vSourceClosure \cup \{v\}$ 
9:     Add  $\overline{C}$  to  $V_{CRG}$ , add  $(vSourceClosure, \overline{C})$  to  $A_{CRG}$ 
10:    Update and store shortest path cost to  $\overline{C}$ 
11:     $ROI_{\overline{C}} \leftarrow \left( Rank(\overline{C}) - Rank(vSourceClosure) \right) / Cost(v)$ 
12:    if An out-neighbour of  $vSourceClosure$  was added then
13:       $maxVertex \leftarrow$  an out-neighbour of  $vSourceClosure$  with maximum ROI
14:       $Process \leftarrow \{maxVertex\}$  {Clear rest of queue}
15:    else
16:       $Process \leftarrow \{v \in V_{CRG} \mid Rank(v) \text{ is maximal}\} - Seen$ 
17:   $maxVertex \leftarrow$  a vertex in  $V_{CRG}$  with highest rank-sum
18:   $RankEliminated \leftarrow$  rank-sum of  $maxVertex$ 
19:   $(PCVS, TotalCost) \leftarrow$  (vertex set, cost) in a least cost path to  $maxVertex$ 
20: return PCVS, TotalCost, RankEliminated

```

4.4.6 Equivalent removals

The closure-relation graph enables us to easily find vertex deletions that have the same closure. To illustrate, we first present Figure 4.5 which gives a randomly generated 10-host network. Notice that two hosts, host0 and host2, are prioritized for defence, and the attacker is located on host4. Consider the case where the budget is only large enough to make one patch. Since host0 has three vulnerable services, patching any one of them will still allow the attacker to gain control of the host by exploiting the other two services. Then, the attacker could also reach host2 by exploiting its vulnerable service. Thus, if the budget is sufficient to only allow one patch to be applied, the correct course of action is to patch the vulnerable service on host2 so that at least one host is protected.

Figure 4.6 is the attack graph for Figure 4.5 and the first vertex deletion suggested by the combination of AssetRank and Algorithm 4.3 is shown. The algorithms correctly suggest that the vulnerable service on host2 be patched because this action will protect the network to the greatest degree, as evidenced by removing the greatest amount of rank from the attack graph.

We have implemented a Python module that computes and visualizes closure-relation graphs. A full closure-relation graph computed by a brute-force approach would be much too large to show but the subgraph explored by Algorithm 4.3 is manageable with a small amount of magnification. Figure 4.7 shows the closure-relation graph for the scenario just discussed and Figure 4.8 gives a magnified section. The figures show the branches computed in order to find a solution for the complete disconnection of the attacker and goal subcommunities. The darkness of the vertex colouring corresponds with the closure rank-sum for that vertex.

The vertex with the maximum closure rank-sum corresponds with a vertex deletion set that maximally separates the source and target subcommunities in the attack graph. The least cost path from the root to the maximum closure rank-sum vertex gives the least-expensive vertex deletion set that the algorithm found. The blue heavy lines indicate vertex deletions that are eventually part of the least cost path.

We can read equivalent removals for vertex deletions from the closure-relation graph. For example, while the highest ROI path from the root vertex is along the arc labelled 23, we see that the arcs labelled 33 and 34 both lead to the neighbouring vertex on the right side. Thus, vertices 33 and 34 result in equivalent deletions for any children of the root vertex and so we save time in subsequent computations by using the fact that the vertices are equivalent and therefore we only need to compute the result of deleting any one of them. In general, AND/OR graph vertex deletions relating vertex x to y form an equivalence class. All vertices on a path from x will share the equivalence classes, and the equivalence classes could combine into larger equivalence classes. This fact is proven in the following proposition.

Proposition 4.2. *If a and b are equivalent in an AND/OR graph G induced by S and T , they are equivalent in every subgraph G' of G (induced by S and T).*

Proof. Let G' be a subgraph of G (both induced by vertex subsets S and T). Then $G' = G - \bar{C}$ for some vertex deletion set C . If a and b are equivalent vertex deletions

in G then,

$$\begin{aligned}
G - \bar{a} &= G - \bar{b} \\
\Rightarrow (G - \bar{a}) - \bar{c} &= (G - \bar{b}) - \bar{c} \\
\Rightarrow (G - \bar{c}) - \bar{a} &= (G - \bar{c}) - \bar{b} \\
\Rightarrow G' - \bar{a} &= G' - \bar{b} \\
\Rightarrow a \text{ and } b &\text{ are equivalent vertex deletions in } G' .
\end{aligned}$$

□

In Figure 4.7, an example of equivalence classes combining are the classes [17] and [20,21]. They are separate equivalence classes in the full attack graph G (represented by the root vertex in the closure-relation graph) where deleting 17 ($G - \overline{\{17\}}$) removes {16,17} and deleting 20 ($G - \overline{\{20\}}$) or 21 ($G - \overline{\{21\}}$) removes {11,12,13,14,15,16,17,18,19,20,21,29,30,37,38}. However, they combine into the same equivalence class [17,20,21] in all subgraphs of $G' = G - \overline{\{23\}}$. In the electronic version of this document, the reader can magnify Figure 4.7 and verify that deleting 17, 20, or 21 all have the same effect in the subgraphs G' , $G' - \overline{\{62\}}$, and $G' - \overline{\{62,44\}} = G - \overline{\{23,62,44\}}$, because they form the equivalence class [17,20,21].

4.4.7 Redundant deletions

Algorithm 4.3 might return a vertex deletion set with redundant removals. This will happen if some elements of the set that are removed initially are contained in the closure of a disjoint subset of the vertex deletion set. That is, for a vertex

deletion set C , indexed by I , there exists an $i \in I$ where:

$$C_i \cap \overline{\bigcup_{i \neq j} C_j} \neq \emptyset \quad (4.7)$$

For example, if there are vertices a, b in the deletion set and if a is removed first, followed by b but $a \in \overline{b}$, then $\{a, b\}$ is a suboptimal set that could be returned by the BFS_PCVS algorithm. This situation can be corrected by using a hybrid algorithm that runs Algorithm 4.2 after running Algorithm 4.3 and rather than considering all vertices when computing the deletion set, it only considers combinations of the vertices returned by Algorithm 4.3. In that case, the number of vertices to consider is greatly reduced and so the exponential complexity of Algorithm 4.2 is feasible since a relatively small number of vertices will be considered (*e.g.*, tens of vertices).

Another example is the vertex deletion set shown in Figure 4.7 ($\{23, 62, 44, 53\}$). The vertex deletions were found in the order they are presented and the set corresponds with patching the service on host2 and all 3 services on host0. However, if the budget is sufficient to patch all three services on host0, then the patch on host2 is no longer required to prevent an attack from host4. When the hybrid algorithm is asked to compute an optimal solution considering only vertices 23, 62, 44, and 53, it correctly determines that deleting vertex 23 is redundant and returns the solution $\{62, 44, 53\}$.

Removing vertices from the vertex deletion set frees up their cost. If partial connectivity between the source and target sets in the graph exists, the remaining budget is used with Algorithm 4.3 to attempt a better solution. The alternation of Algorithms 4.3 and 4.2 is repeated until no vertex deletions remain within budget or a full solution is found.

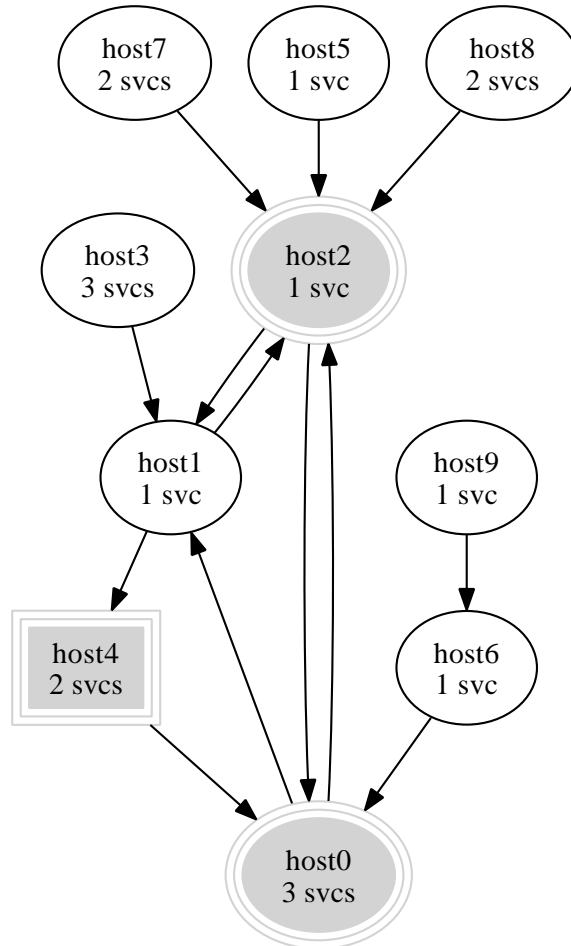


Figure 4.5: Randomly generated scale-free computer network with 10 hosts. The attacker starts at host 4 and the target hosts are 0 and 2.

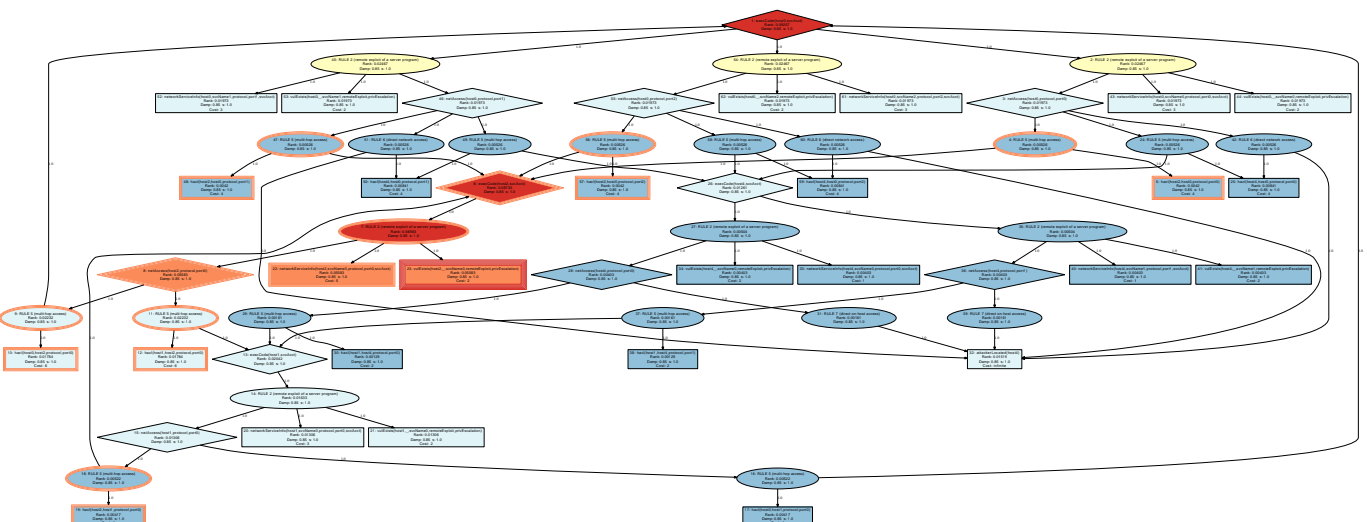


Figure 4.6: Attack graph for network in Figure 4.5. Source vertices are 1 and 6; target vertex is 32. Removing vertex 23 removes all triple border vertices. Redder (bluer) vertices denote higher (lower) AssetRank values. Ellipses, diamonds, and rectangles represent AND, OR, and SINK vertices (respectively).



Figure 4.7: Closure-relation graph for attack graph in Figure 4.6. Vertices are labelled with the closure of the incoming paths and arcs are labelled with the cut vertex and its cost. Darker vertices have a higher sum of closure rank. The figure may be enlarged in the electronic version of this document; a magnified section is shown in Figure 4.8.

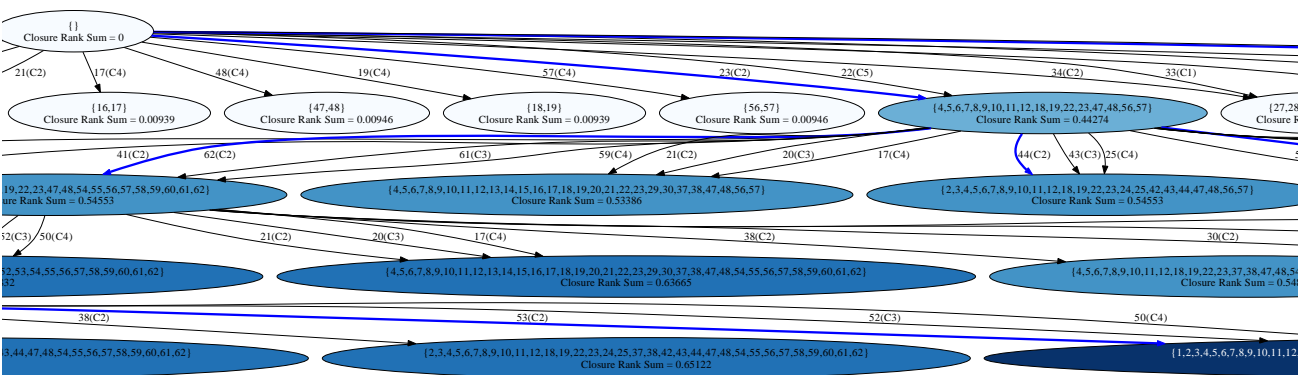


Figure 4.8: Magnified section of Figure 4.7. Blue heavy arcs indicate vertex cuts that are included in the vertex cut set. Vertices that have out-arcs are on the exploration path that leads to the solution.

4.5 Complexity

4.5.1 Vertex set closure

The main work in our approach is done in Algorithm 4.1 (VertexSetClosure). Lines 2 and 3 compute the reachable vertices from a specific set. The algorithm used is a depth-first-search implementation provided by the Python NetworkX module [31]. The implementation only computes reachability from a single vertex so it was modified to compute the reachability of a set of vertices simultaneously. The complexity of the NetworkX reachability algorithm is $\mathcal{O}(v + a)$ where v is the number of vertices in the graph and a is the number of arcs. Since there are no loops in the algorithm, the complexity is $\mathcal{O}(v + a)$. Algorithm 4.1 (VertexSetClosure) is called by the two partial cut-set algorithms.

4.5.2 Brute-force partial-cut vertex set

Algorithm 4.2 (BruteForcePCVS) computes the optimal partial cut-set for the specified budget by a brute-force comparison of all candidate combinations. While the power set of the sink set is of size 2^s where s is the number of sinks, the power set contains many combinations of sinks whose total cost exceeds the budget, so those are eliminated. The size of the remaining set can be much smaller than 2^s . For example, if every SINK has a cost of 1, the budget is b , and every vertex has a positive (non-zero) rank, then the number of candidate vertex cuts is given by $\sum_{k=0}^b C_k^s$ where $C_k^s = s! / (k!(s-k)!)$. As the AND/OR digraph grows in size, the discriminator of a limited budget significantly reduces the candidate space. We have implemented (the details are omitted) an efficient dynamic programming method of generating only the candidate courses of action within budget.

The most complex line in the **for** loop is line 4, which has a complexity of $\mathcal{O}(v + a)$, as shown in Section 4.5.1. Given that the line is executed up to 2^s times the complexity of the algorithm is $\mathcal{O}(2^s(v + a))$ in the worst case of an unlimited budget; however, the algorithm runs within $\sum_{k=0}^b C_k^s$ iterations when a limited budget is specified. A practical method of specifying a budget is to use the solution computed by Algorithm 4.3 (BFS-PCVS) in polynomial time, thus giving a budget upper bound.

One might expect that presorting the COA sets in ascending-cost order would improve performance because the budget limits the number of possibilities to consider, but the sorting process, which is $\mathcal{O}(N \log N)$ where $N = 2^s$ is the number of items to sort, is time-consuming and leads to worse times overall. Note that space efficiencies may be gained by discarding more expensive solutions within the loop instead of waiting until step 11. Our Python implementation generates the candidates as they are used and discards more costly solutions immediately, so the space requirements are minimal.

4.5.3 Best-first search partial-cut vertex set

In Algorithm 4.3 (BFS-PCVS), the highest time complexity is the computation of the closure at line 8. Also, it consumes the most time since it is inside a doubly-nested loop. The **for** loop at line 7 is evaluated over the remaining in-budget vertices found in the (shrinking) AND/OR directed graph given by $V_{AG} - \text{vSource-Closure}$. AND/OR directed graphs directly correspond to logic formulas where the sink vertices are the variables. In that context, the goal is to determine which variables should be made false in order to falsify the portion of the graph corresponding with the greatest rank-sum. Recall that in an attack graph, the sinks represent

facts about the network and they are the only items that can be changed to alter the graph (improve the security of the network). The number of vertices whose cost is within budget is denoted by N in the following discussion. In attack graphs, N is at most the number of sink vertices, which is clearly less than the number of vertices in the graph. The **while** loop at line 3 is executed until no vertices remain within budget due to statement 12. The number of vertices decreases by at least 1 in each iteration¹¹. Line 11 requires a summation where the number of operands is bounded by $|V_{AG}|$.

In summary, lines 3 and 7 each loop up to N times, the complexity of line 8 is $\mathcal{O}(|V_{AG}| + |A_{AG}|)$, giving an overall complexity of $\mathcal{O}(N^2(|V_{AG}| + |A_{AG}|))$. Note that the graph itself is shrinking (line 7) and thus the remaining vertices to check so using the initial N , $|V_{AG}|$, and $|A_{AG}|$ gives a conservative estimate.

4.5.4 Provable goodness bounds of approximation algorithms

The nonlinear knapsack research community gives insight into the problem of computing partial-cut vertex sets. Kellerer *et al.* [45, p. 350] point out that the quadratic knapsack problem¹² is strongly NP-hard by reduction from the clique problem. The supermodular knapsack problem reduces to the problem of computing partial-cut vertex sets since the profit of the combinations of items given by the function Q forms a partially ordered set and so can be represented by a Hasse diagram. The Hasse diagram is a directed acyclic graph where the arrows can be

¹¹The supermodular characteristic of the graph means that the number of vertices decreases rapidly.

¹²The quadratic case is a specialization of the supermodular case (or rather, the supermodular case generalizes the quadratic case) by restricting Q to be of the form $Q(C) = C^T P C$ where P is a non-negative square matrix of order n [27]. Also, Nemhauser *et al.* [61, p. 276] show that non-negative P is a necessary and sufficient condition of supermodularity, so the given representation covers all cases of quadratic supermodularity.

oriented so that $Q(\mathbf{1})$ is the source community S and $Q(\mathbf{0})$ is the target community T . The weight of the knapsack items are the cost of vertex removal, the cost of all other vertices (combinations and $Q(\mathbf{0})$) are set to infinity. The incremental profit of each Hasse vertex corresponds to the rank metric. The maximum weight of the knapsack is the partial-cut budget. Thus, computing partial cuts is at least as hard as the supermodular knapsack problem, and so is NP-hard.

Korte and Schrader [47] show that unless $P = NP$, no fully polynomial-time approximation scheme (FPTAS) exists that gives an approximation to the exact solution for strongly NP-hard problems in time polynomial in both the problem size and inversely to the goodness bound (*i.e.*, $1/\epsilon$). However, NP-hard problems arise often in practical applications and finding good solutions to them is important work. Bang-Jensen and Gutin discuss the point of finding practical solutions to hard problems in their book on Digraphs:

In the rest of this section we describe methods that do not give us any guarantee on the quality of the solution and sometimes not even on the running time of an implementation of the method. But experimental evidence suggests that in practice some of these so-called heuristics do give solutions which are close to the optimum solution. Furthermore, they often run very fast when implemented carefully on a PC. Such methods may not seem very interesting to the theoretician who may only consider methods that provably obtain the optimum or some approximation guarantee for the solution as worth studying. However, in practice the situation is entirely different: the engineer who has been asked to find a reasonable solution to an instance of the Feedback Arc Set problem, say, cannot really use this attitude. What (s)he needs is a way to get a good solution and some indication that this solution is better than a random solution and cannot be easily improved on. Certainly such a solution will often be much better than one that could be found at hand by the engineer. [7, p. 673]

4.6 Summary

We introduced the *closure* of a vertex deletion set in the context of AND/OR directed graphs and presented a polynomial time algorithm that leverages rank and loss cost research. We applied the work to the computation of compound course-of-action recommendations that maximally disrupt attackers, within a specified budget. Our examples demonstrate that practical solutions can be found by the polynomial-time best-first search greedy algorithm. Our algorithm makes effective recommendations for improving security even when practical considerations prevent the network from being completely secured.

It is possible to use the greedy and brute-force algorithms cooperatively in order to achieve better results. The greedy algorithm, because it removes the single most effective SINK vertex in each iteration, can sometimes expend part of the budget removing a vertex that will itself be indirectly removed in a later iteration. This results in a vertex deletion set that is a superset of an equivalently good solution. Such supersets can be compressed by applying the brute-force algorithm to them. These sets are usually small so the brute-force algorithm can execute quickly, and when it succeeds in reducing the partial-cut vertex set (and its associated cost) the greedy algorithm can resume searching with a larger remaining budget. This optimization can be computed very quickly and we recommend the hybrid approach.

Our technique can be applied to attack graphs in both proactive and reactive scenarios. The difference between the two situations is the range of actions, and the cost of implementing the actions. In the next chapter we show the integration of the algorithms into an automated computer network defence system.

Chapter 5

Experiments

5.1 AssetRank experiments

In this section we present several experiments we conducted to study 1) AssetRank's efficacy in giving results consistent with the importance of an attack asset to a potential attacker; and 2) how the AssetRank metric may be used to better understand security threats conveyed in a dependency attack graph, as well as in choosing appropriate mitigation measures.

In our experiments, we use the MulVAL attack-graph tool suite to compute a dependency attack graph based upon a network description and a user query. For example, a user may ask if attackers can execute code of their choosing on any server. The attack graph is exported to a custom Python module. The Python module normalizes the input data, computes the AssetRank values, and visualizes the attack graph using the graph visualization software Graphviz [22].

5.1.1 Algorithm evaluation plan

To evaluate the effectiveness of the ranking algorithm we have performed the following experiments:

1. Demonstrate the effect of arc weights and vertex-specific damping factors on a 3-computer network since it is small enough for a human to anticipate the parameters' effects. Arc weights are demonstrated by comparing a vulnerability with a 0.4 likelihood of successful exploit with a vulnerability having a 0.8 likelihood. The ranks of the vulnerability, service, and connectivity connected to the 0.8 likelihood vulnerability should be ranked higher than the lower probability vulnerability. Damping factors are evaluated in a similar manner.
2. Examine the ranks in a 5-computer network that has more complex network routing, but a scenario that is still simple enough to judge whether the ranks correspond with what subject matter experts consider to be appropriate.
3. Finally, the ranks are evaluated in a network that is based on a production control-system network. The attack graph in this scenario is too complex for a human to fully comprehend and predict the correct ranks but once the ranks are computed and analyzed, a subject matter expert can recognize the reasonableness of the ranks as a measure of the importance of the corresponding network facts. That is, it is practical for a subject matter expert to verify the solution, but extremely challenging to construct the solution.

5.1.2 Experiment 1: Arc weights and damping factors

The first experiment demonstrates the effect of arc weights and vertex-specific damping factors on a small network. Figure 5.1 shows the network for experiments 1a and 1b. The attacker has access to both PC1 and PC2. User1 is on PC1, which has vulnerability Vul1, and User2 is on PC2, which has vulnerability Vul2. PC1 and PC2 have access to the defence-priority goal computer but not to each other.

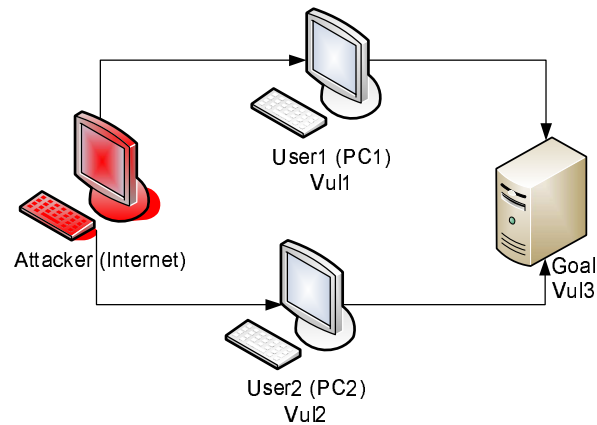


Figure 5.1: Scenario for experiments 1a and 1b

In Experiment 1a we show how the rank values can take into account the tool maturity for exploits by using the arc weights. We assume that Vul1 has functional exploit tools available and Vul2 has only proof-of-concept code available; hence, we assign success likelihood metrics of 0.8 and 0.4, respectively. A uniform damping factor of 0.99 is applied to all vertices, which has the effect of pushing criticalities to the perimeter devices in the network. We expect that Vul1 will have a higher rank metric than Vul2 since the attacker is more likely to prefer it. Figure 5.2 shows the attack graph coloured according to the assets' AssetRank values. Vertices represent assets (or capabilities) that the attacker can use to further his

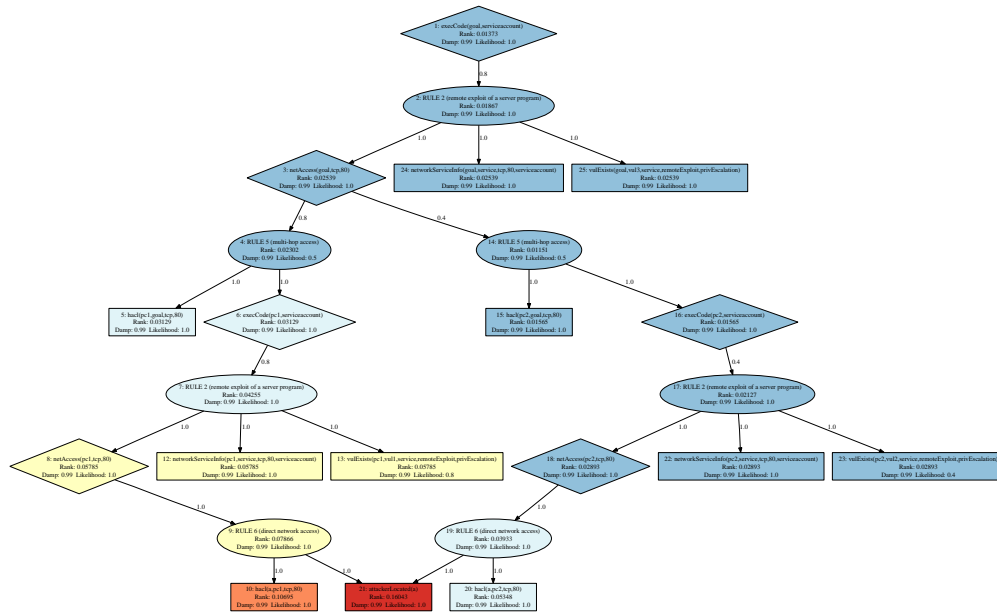


Figure 5.2: Ranked attack graph for the Experiment 1a scenario. Vertex colours range from blue to red with blue indicating vertices with relatively lower ranks and red indicating vertices with higher ranks.

attack, and arcs indicate dependencies that each derived asset has on other assets. In MulVAL, a tuple $\text{vulExists}(\text{Host}, \text{VulID}, \text{Account}, \text{AccessVector}, \text{Consequence})$ means “device Host has the vulnerability VulID in software running as Account that is exploitable via AccessVector with the result Consequence.” A tuple $\text{hacl}(H1, H2, \text{Protocol}, \text{Port})$ means “host H1 can reach host H2 via Protocol and Port.” The graph gives a proof tree for how the asset $\text{execCode}(\text{goal}, \text{serviceaccount})$ is derived from other facts. Table 5.1 shows the rank metrics for the two vulnerabilities Vul1 and Vul2. Our algorithm computes a value of 0.0579 for Vul1 and a value of 0.0289 for Vul2 which is consistent with the higher value that Vul1 has to the attacker.

In Experiment 1b we show how AssetRank can account for out-of-graph de-

Attack asset	Rank
vulExists(pc1,vul1,service,...)	0.0579
vulExists(pc2,vul2,service,...)	0.0289

Table 5.1: AssetRanks for Experiment 1a

dependencies through vertex-specific damping factors. We assign both Vul1 and Vul2 a success likelihood of 0.5 so that we can focus our attention on an out-of-band social engineering attack. To that effect, we assume that it is 80% likely that PC1 will be compromised by ways not shown by the attack graph (for example, obtaining User1's log-in credentials through social-engineering), and PC2 is 40% likely to be compromised in such ways. These values correspond with User2 having received more training and so being more security-vigilant than User1. In this case, we expect that Vul1 will be ranked lower than Vul2 since the attacker has a lower dependence upon it. Figure 5.3 shows the attack graph coloured according to the assets' rank values and Table 5.2 shows a selected portion of the vertices and their scores. As we can see, Vul2 has a score of 0.0414 and Vul1 has a score of 0.0310. This ranking is intuitively correct since attackers have a greater chance of obtaining PC1 without exploiting its vulnerability, so Vul1 is less important to them.

Attack asset	Rank
vulExists(pc1,vul1,service,...)	0.0310
vulExists(pc2,vul2,service,...)	0.0414

Table 5.2: AssetRanks for Experiment 1b

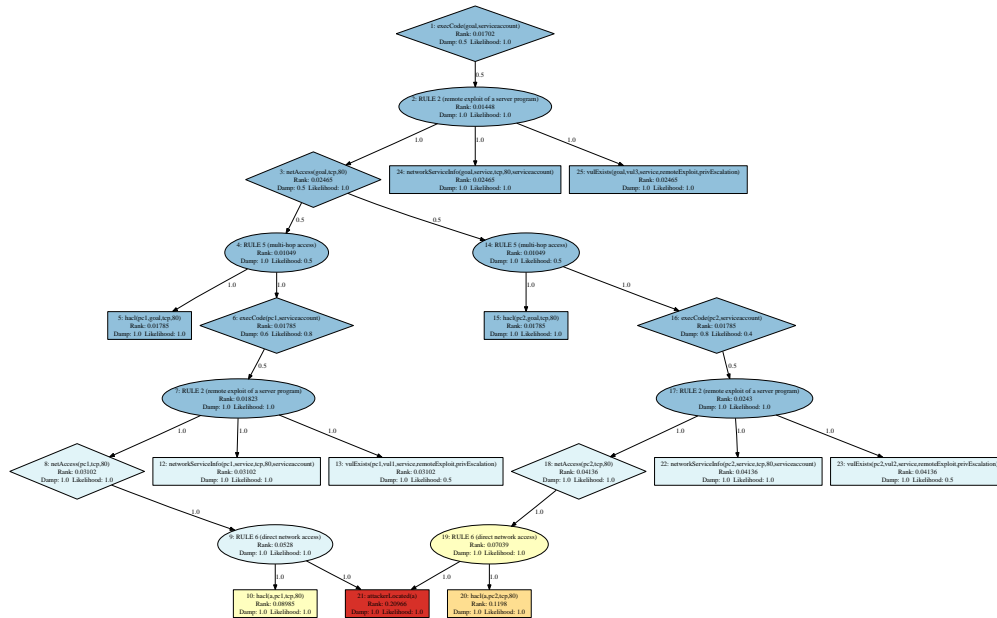


Figure 5.3: Ranked attack graph for the Experiment 1b scenario

5.1.3 Experiment 2: 5-host network

Now that we have shown the effect of arc weights and vertex damping factors, we demonstrate the results of applying AssetRank to the attack graph for the slightly more complicated example network in Figure 3.1. In the first scenario, we assume all the vulnerabilities have the same level of difficulty to exploit, represented by identical success likelihood metrics.

A portion of the resulting ranking is shown in Table 5.3, and the complete attack graph with coloured vertex ranking can be found in Figure 5.4. Host Access Control List (HACL) tuples are high-level abstractions of the effects of network traffic-control devices such as firewalls, routers, and switches, whose settings a system administrator can modify. The ranking of the HACL predicates demonstrates the effectiveness of AssetRank. The removal of the first HACL predicates

Attack asset	Rank
hacl(e,f,tcp,80)	0.0267
hacl(a,d,tcp,80)	0.0240
hacl(a,c,tcp,80)	0.0240
hacl(d,e,tcp,80)	0.0167
hacl(c,e,tcp,80)	0.0167
hacl(a,b,tcp,80)	0.0129
vulExists(c,vulid2, ...)	0.0323
vulExists(d,vulid1, ...)	0.0323
vulExists(e,vulid4, ...)	0.0274
vulExists(f,vulid5, ...)	0.0219
vulExists(b,vulid1, ...)	0.0174

Table 5.3: AssetRanks for Experiment 2a

eliminates the only path to F. The next two HACL predicates represent the connectivity from the attacker (A) to either of the two hosts (C or D) that can reach into the right subnet. The fourth and fifth HACL predicates represent the network connections from C and D to the only computer in the right subnet they can reach (E). The sixth HACL predicate shows the attacker's ability to reach B, thus giving the opportunity to gain a foothold inside the network but since B cannot directly reach the right subnet, the next step requires exploiting either C or D. We see that the AssetRank values correctly reflect the importance of network routes to an attacker, and thus identify the routes whose removal would be most effective in protecting computers in the network. All computers in this example were given equal personalization value, which corresponds to the goal of protecting as many computers as possible. The AssetRank values for the HACL tuples are consistent with this goal and the intuitive importance of the various attacker assets. Likewise, the vulnerabilities on C and D are the most valuable to the attacker since they allow

him to reach the largest number of hosts.

Now suppose the vulnerability `vulid2` on computer C is very difficult to exploit, and the other vulnerabilities are easy to exploit. We therefore assign the metric 0.2 to `vulid2` and a metric of 0.8 to the other vulnerabilities. The AssetRanks of the new configuration are given in Table 5.4 and the full coloured attack graph is in Figure 5.5.

Attack asset	Rank
<code>hacl(a,d,tcp,80)</code>	0.0406
<code>hacl(d,e,tcp,80)</code>	0.0304
<code>hacl(e,f,tcp,80)</code>	0.0287
<code>hacl(a,b,tcp,80)</code>	0.0168
<code>hacl(a,c,tcp,80)</code>	0.0097
<code>hacl(c,e,tcp,80)</code>	0.0076
<code>vulExists(d,vulid1, ...)</code>	0.0453
<code>vulExists(e,vulid4, ...)</code>	0.0303
<code>vulExists(f,vulid5, ...)</code>	0.0229
<code>vulExists(b,vulid1, ...)</code>	0.0188
<code>vulExists(c,vulid2, ...)</code>	0.0127

Table 5.4: AssetRanks for Experiment 2b

What is remarkable in the new ranking is that the vulnerability on computer C is ranked much lower than before, since it is hard to exploit and an easier route exists. Now computer D becomes much more valuable to the attacker since it is likely to be the only feasible stepping stone into the right subnet, which is manifested by the boosted values on both the vulnerabilities and reachability relations involving D. Note that the vulnerability on computer B is the same as the one on computer D. But since B only indirectly helps the attacker penetrate deeper into the network, its vulnerability's rank is lower than that of D.

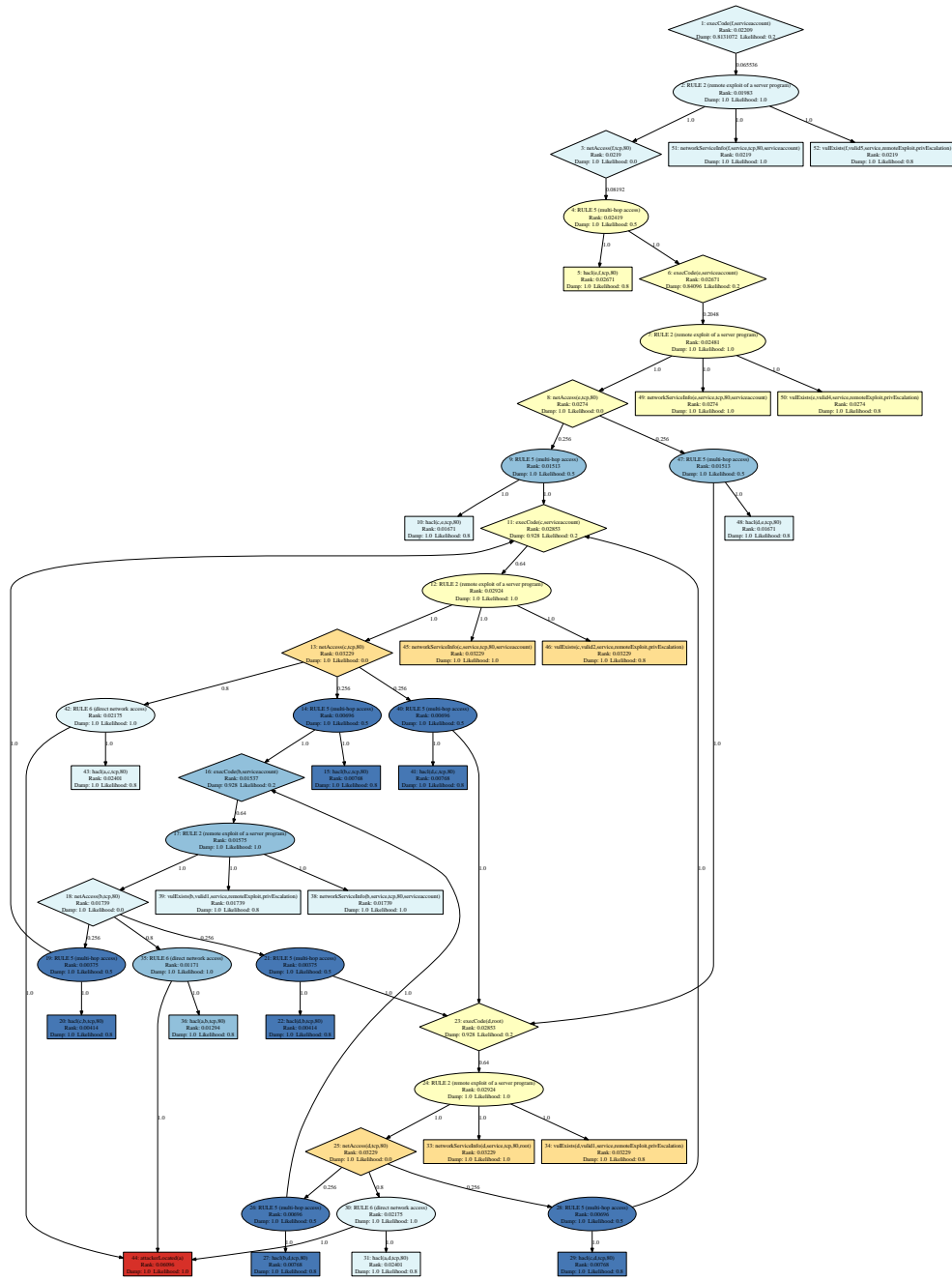


Figure 5.4: Ranked attack graph for the Experiment 2a scenario

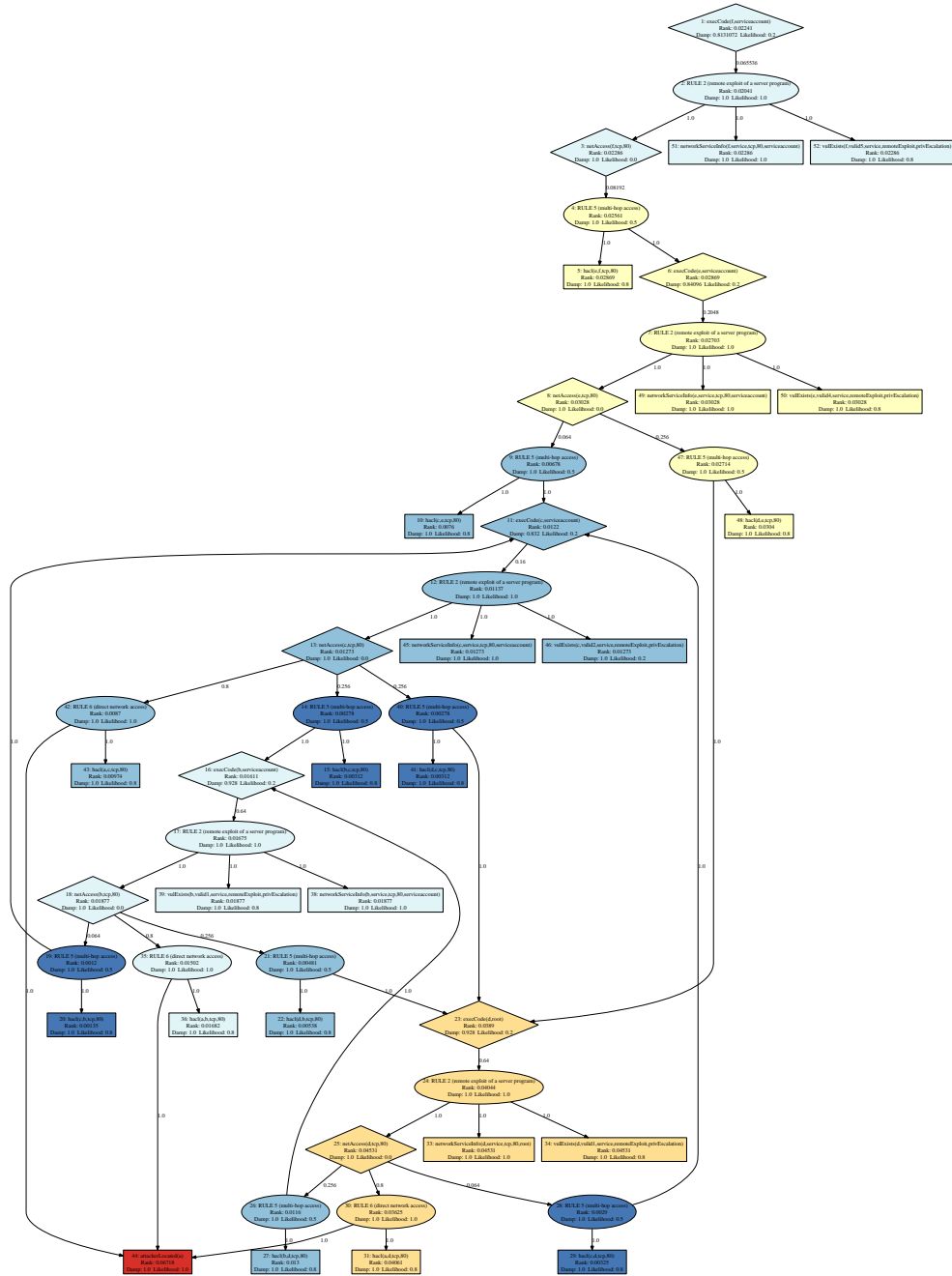


Figure 5.5: Ranked attack graph for the Experiment 2b scenario

5.1.4 Experiment 3: Control-system network

To study how AssetRank works in a more complicated realistic setting, we tested it on a network scenario adapted from a real control-system network, shown in Figure 5.6. The network is protected by a firewall from the Internet. Only computers in the DMZ subnet can be directly accessed from the Internet zone. The computers in the CORP internal subnet can freely access the Internet. Only one computer in the network, the Citrix server, can access the control-system subnet (the Energy Management System, or EMS) that is protected by another firewall, and it may only access the Data Historian. Assuming the attacker is on the Internet and wants to obtain privileges on the Communications Servers in the EMS subnet, there are two obvious entry ways for him: the web server and the VPN server, both of which can be directly accessed from the Internet.

We introduced hypothetical vulnerabilities in this scenario and assigned metrics for them based on our understanding of typical security problems in this type of network.¹ We applied AssetRank on this example and the resulting coloured attack graphs can be found in Figure 5.7. The ranking identifies the two most critical vulnerabilities in the network. One is a remote buffer overflow vulnerability on the web server, which would allow a remote attacker to gain code execution privilege in the DMZ subnet. The other is a browser vulnerability on the user workstation. Since outbound traffic from the CORP Internal zone is not restricted, an unsuspecting user may browse to a malicious website and compromise his computer. This compromise will yield privileges on the internal network to the attacker. There are many other vulnerabilities in the network and there are

¹In non-simulated applications, this information will automatically be furnished by data collection agents installed on the computers and the CVSS metrics provided by the NVD.

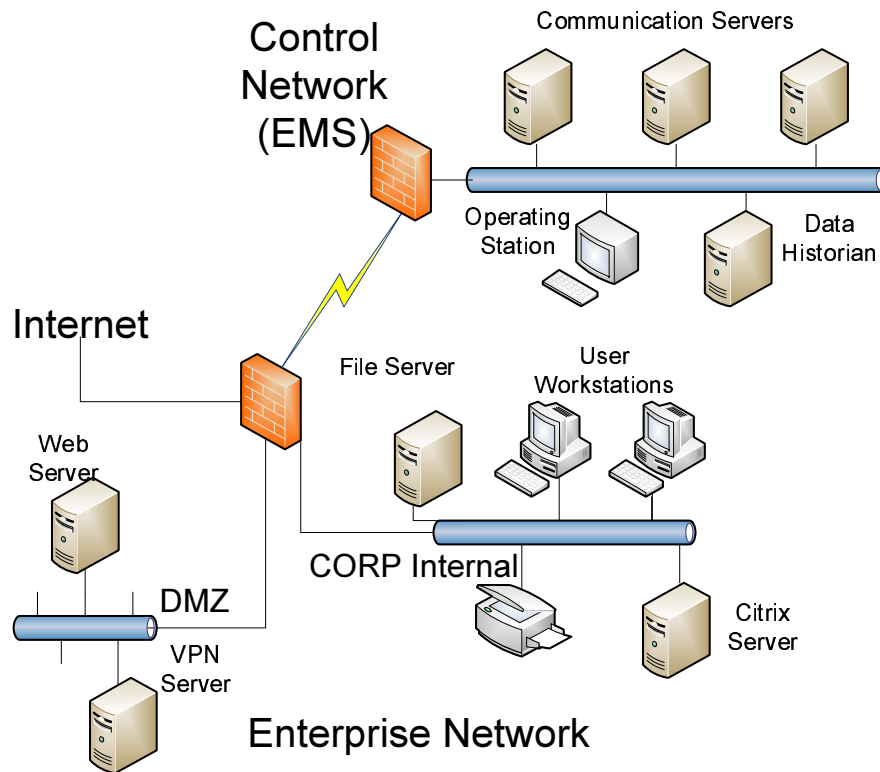


Figure 5.6: A realistic network scenario for Experiment 3

other ways to penetrate into the system (for example, through the VPN server). But the two critical problems identified by the AssetRank algorithm are consistent with a subject matter expert's conclusion after spending an extensive amount of time studying the information revealed by the complicated 129-vertex attack graph with 185 dependencies.

5.1.5 Discussion

A very useful aspect of AssetRank in the context of attack graphs is to assist in prioritizing further analysis and understanding of the threats. We have used the

AssetRank values to colour the attack graph vertices so that a user's attention is immediately focussed on the most critical portion. The lowest ranked vertices are coloured blue and the highest ranked vertices are coloured red. This colouring is intended to be analogous to water faucets where the hot (and dangerous) tap is coloured red and the cold tap is coloured blue. The values could also be used to incrementally show the vertices in an attack graph, with the highest ranked vertices shown first followed by the lower-ranked ones.

We have shown that arc weights are a flexible instrument that allow the user to take attacker preferences into account. We used the weights to favour attacks with mature exploitation techniques over unproven attacks. Alternatively, the metric can be used to denote other attack characteristics or a combination of them, including:

- Stealthiness of an attack — allows the inclusion of IDSs in the model by giving a penalty for attacks leaving evidence (log entries or system crashes for example) or detectable attacks over links monitored by an IDS.
- Resources required — gives the ability to penalize resource consuming attacks (for example, attacks that require password cracking or large bandwidth).

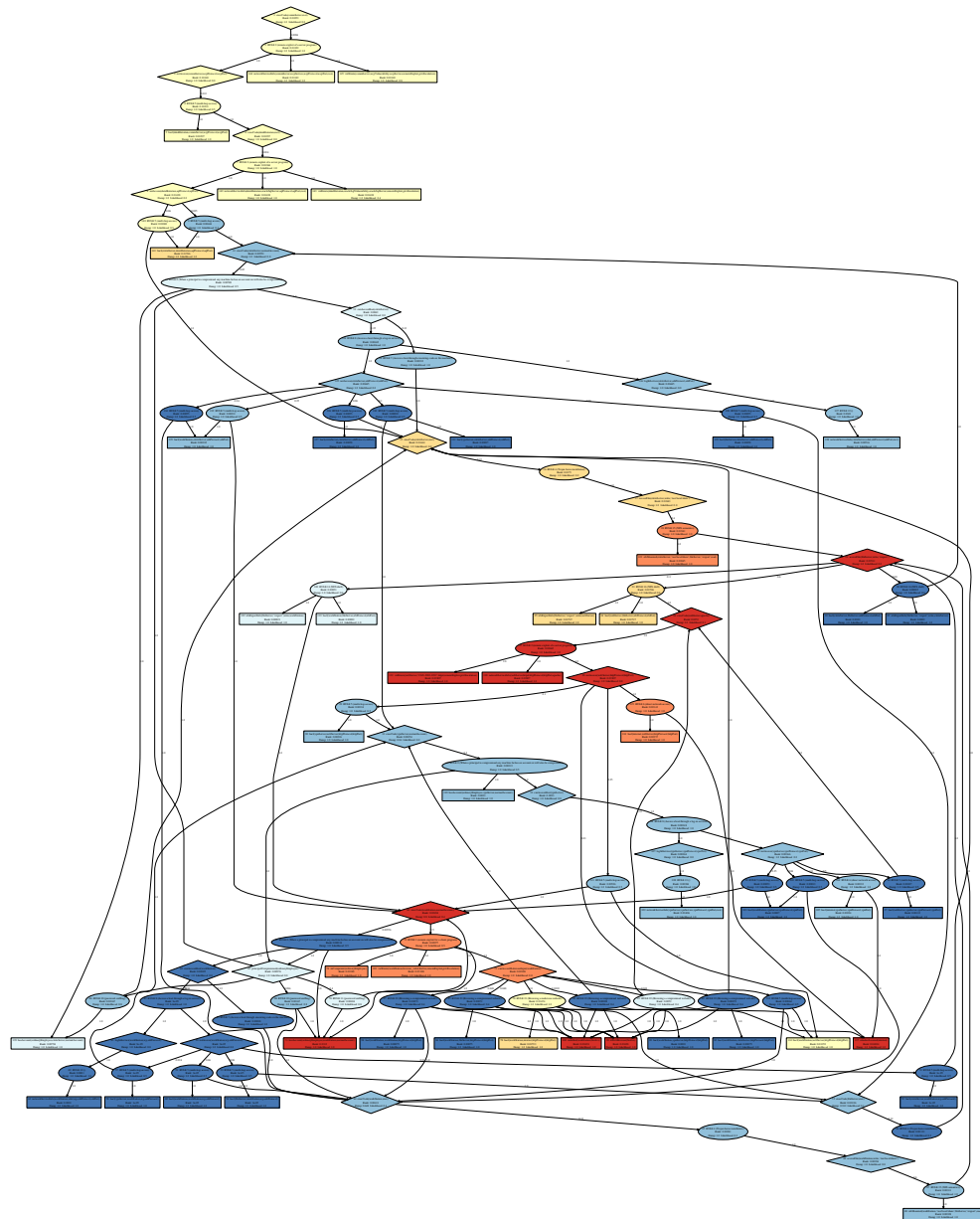


Figure 5.7: Ranked attack graph for the Experiment 3 scenario

5.2 Partial-cut vertex set experiments

We illustrate the ability of our partial-cut vertex set algorithms to provide decision support to computer network security by generating courses of action.

5.2.1 Algorithm evaluation plan

A proof sketch for the total correctness of the brute-force algorithm was given in Proposition 4.1 for the exponential-time brute-force partial cut-set algorithm, showing that it always terminates and that the output is correct with respect to the specification. Our first experiment demonstrates the application of the brute-force algorithm. The network and attack graph are small enough to enable a full understanding of the concepts.

To evaluate the effectiveness of the polynomial-time partial-cut vertex-set algorithm, which uses a best-first search (hill-climbing) strategy, we compare it to the exponential-time brute-force algorithm on thousands of networks generated using preferential attachment. The accuracy of the polynomial-time algorithm is compared with the exponential-time algorithm for networks of various sizes with the polynomial-time algorithm giving the optimum answer 99% of the time.

Finally, we present an integration of commercial software, the open source attack graph generator, our ranking algorithm, and our partial-cut vertex set algorithms. The integration is tested on a live 9-host network to demonstrate the use of the algorithms in a computer network defence application. The network is representative of an organization with several servers and network zones.

5.2.2 Experiment 1: Simple 3-host network

The network in Figure 5.8 has three hosts plus the attacker's PC. Each computer has a remotely exploitable vulnerability. Public databases give information regarding the nature of particular vulnerabilities, and the maturity of attacks against them. Given this information, we set the likelihood of an attacker successfully exploiting the vulnerability on the web server to 80%, the mail server to 40%, and the file server to 100%. The defender's primary concern is safeguarding the file server. The attacker does not have direct access to the file server but can reach it by a multistep attack through either the web server or the mail server. This scenario is closely related to the first AssetRank example presented in Figure 5.1. The main difference is that the two intermediate hosts have network access to each other.

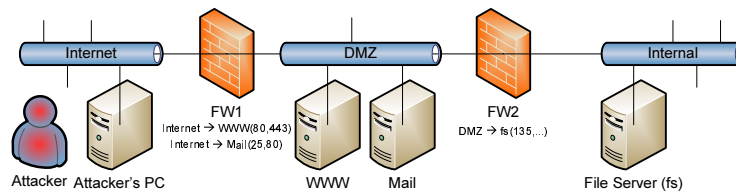


Figure 5.8: Example 1 network: The attacker can reach the file server by a multistep attack through the web server or mail server.

Figure 5.9 is the attack graph containing the possible attacks for this scenario; it contains a proof for how the asset `execCode(fs, system)` is derived from other facts. The vertices related to the web server have higher rank than those related to the mail server because the exploit on the web server has a higher likelihood of success.

In a network as small as this example, the defender will likely have the resources to patch all software; however, we assume enterprise-level resource re-

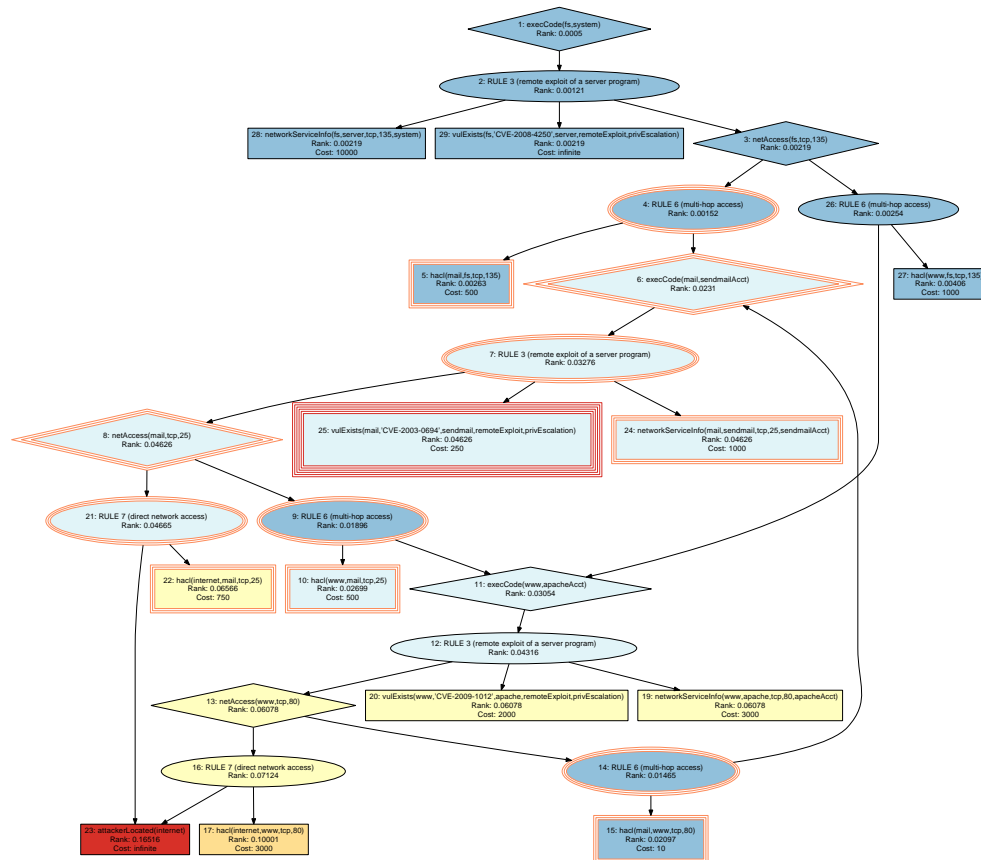


Figure 5.9: Ranked attack graph for the Example 1 network: The graph contains a proof of the attacker’s ability to obtain code execution privileges on the file server. Vertex 25 (decorated with 6 peripheries) is the recommended COA under loss function “Cost 2” with a budget of 1000. Removing vertex 25 corresponds with patching the mail server vulnerability. When vertex 25 is removed from the graph, the rest of its closure (indicated by the vertices with 3 peripheries) is also removed since those vertices are no longer useful to the attacker in reaching the defence goal. The closure of vertex 25 contains 39% of the graph rank.

restrictions to readily demonstrate the benefits of COA analysis in a network small enough to evaluate. Table 5.5 gives costs under two different loss functions. The column “Rank-sum of closure” gives the sum of the rank weights for the closure

of each vertex. The goal is to deny the attacker the greatest amount of rank; a rank-sum of 1 corresponds with complete disconnection of the goal to defend (`execCode(fs, system)`) and the attacker (`attackerLocated(internet)`). A total disconnection corresponds with completely eliminating the attack. The full graph contains 28 vertices and 30 arcs. The residual vertices and arcs columns give the number of vertices and arcs remaining in the graph after the vertex closure is removed. One immediately notices that the number of residual vertices or arcs is a poor indicator of the solution quality. For example, the removal of either vertex 5 or 17 results in the same number of residual vertices and arcs but the rank-sum removed by 17 is over 40 times higher than that of 5.

The first column (Cost 1) simply assigns a cost of 250 for every vulnerability patch, 500 for every network route removal, and 1000 for every service shutdown. Using column Cost 1 and a budget of 1000, we immediately notice two solutions that remove the entire attack: vertex 28 at a cost of 1000, and vertex 29 at a cost of 250.

Column Cost 2 presents loss function costs from a manual process. Loss functions are an active and difficult area of decision theory research and our scope is limited to the optimization of return on investment for courses of action. The following assumptions are examples of information used by the loss function: an inconvenient workaround is available for the web server vulnerability (vertex 20 – cost 2,000); a well-tested patch is available for the mail server vulnerability (vertex 25 – cost 250); the file sharing service (vertex 28 – cost 10,000) is critical to operations and its availability is the highest priority; a patch does not exist for the file server vulnerability (vertex 29 – cost ∞). Using Cost 2 and a budget of 1000, it is not at all obvious what the best solution is from an examination of the graph or ta-

ble. Figure 5.9 illustrates the solution found by both algorithms BruteForcePCVS and BFS-PCVS; namely, the removal of vertex 25 at a cost of 250. There are several other combinations (*e.g.*: {5,10}, {10, 15, 25}, {22}, *etc.*), but every vertex within the budget is contained in the closure of vertex 25, so it is the optimal COA, removing the greatest amount of rank (39%) from the graph. The least cost COA that fully removes the attack on the file server is {25,27} for a cost of 1250.

Vertex		Rank-sum of closure	Residual		Cost	
number	Description		Vertices	Arcs	1	2
5	route: mail to fs	0.00415	26	27	500	500
10	route: www to mail	0.04595	26	27	500	500
15	route: mail to www	0.03562	26	27	500	10
17	route: internet to www	0.17125	26	27	500	3000
19	apache service on www	0.51546	15	14	1000	3000
20	vulnerability on www	0.51546	15	14	250	2000
22	route: internet to mail	0.11231	26	27	500	1000
23	attacker exists	1	0	0	∞	∞
24	sendmail service on mail	0.39267	15	14	1000	1000
25	vulnerability on mail	0.39267	15	14	250	250
27	route: www to fs	0.0066	26	27	500	1000
28	server service on fs	1	0	0	1000	10000
29	vulnerability on fs	1	0	0	250	∞

Table 5.5: Vertex costs for Figure 5.9 graph

5.2.3 Experiment 2: Scalability and accuracy

In this section we demonstrate the scalability and efficacy of the Best-First Search algorithm, compared to the brute-force algorithm. To do this, we randomly generate networks comprised of 10, 100, and 1000 hosts using the process described in Section 4.2. One may think of each configuration as a subnet that contains multiple hosts but all hosts in the subnet share a common baseline configuration. For the purpose of generating a MulVAL attack graph, this format is sufficient since

MuVAL is not concerned with the actual number of hosts, only the number of different configurations.

Graph generation

First, a directed scale-free graph is generated where each vertex of the graph represents a baseline host configuration. Each host configuration contains between 1 and 3 remotely exploitable vulnerabilities. Since the network is formed using preferential attachment, a large number of host configurations do not offer connectivity for incoming connections. Hosts sharing a baseline configuration are assumed to be in the same subnet and have unrestricted connectivity to other hosts in the subnet.

Next, an attack graph is generated for the computer network. The attacker's starting location is chosen randomly. Most often, the attacker will have no incoming connections but occasionally it will be a host configuration with dense connections.

The number of attack targets varies from 1 to 10% of the number of host configurations. The targets are selected by choosing the hosts with the highest in-degree (most incoming connections). Since the attacker starting location is chosen randomly, care is taken to avoid choosing the attacker's location as a target.

Course of action parameters

A loss function cost is associated with each course-of-action as follows:

- Removing a network connection from host A to B costs the number of in-neighbours of A, plus 1 for A itself. The rationale is that host A depends

on host B for functionality. Removing the connection between A and B will adversely affect all of the hosts connecting to A.

- Shutting down a service on host A costs the number of in-neighbours of A. The same rationale is applied here as was applied for network connections except that no cost is added for A itself.
- Patching a vulnerability is a constant cost of 2. In future work, a more complex cost that takes into account downtime, baseline testing, and sometimes unavailable patches could be considered. However, a cost of 2 will allow for a diversity of course of action selections to be returned by the algorithms since it will be less expensive to cut connectivity or shut down a service when only one host relies on the service.

For each attack graph, the COA budget is incremented from 1 to 25% of the number of hosts in increments of $\lceil 0.25(\text{num hosts})/10 \rceil$ to give up to 10 steps.

Algorithm comparison

Since the brute-force algorithm (BruteForcePCVS) is too slow to be applied to networks with 1,000 hosts, it was only applied to networks with 10 and 100 hosts. The Best-First Search (BFS) algorithm was applied to all three sizes of networks over various combinations of random networks, number of goals, and budget levels. The BFS algorithm gives the same answer as the brute-force algorithm 99% of the time in networks with 10 and 100 hosts, but with a much shorter computation time. Table 5.6 presents the timing and accuracy results, while Table 5.7 gives statistics for the solution ratios.

Property	10 hosts	100 hosts	1000 hosts
Combinations (network, goals, budget)	1504	2622	108
Brute-force computing time range (sec)	1.08 – 87.87	1.14 – 50,652.95	N/A
BFS computing time range (sec)	0.00 – 0.16	0.00 – 1.95	0.01 – 85.67
Cases with different COA total rank	14 (0.9%)	28 (1.1%)	N/A

Table 5.6: Comparison of the Brute-Force and Best-First Search algorithms over various combinations of random networks, number of goals, and budget levels.

Hosts	Count	Min	Max	Average	Standard Deviation
10 hosts	14 of 1504	0.660	0.916	0.780	0.058
100 hosts	28 of 2622	0.770	0.990	0.938	0.049

Table 5.7: Ratio statistics for partial-cut solutions where the BestFirstSearchPCVS result is non-optimal (BruteForcePCVS always produces optimal solutions). The rank-sum of the solutions produced by the BestFirstSearchPCVS algorithm are divided by those produced by the BruteForcePCVS algorithm, giving the ratio.

Our experiments demonstrate that practical solutions can be found by the polynomial-time best-first search greedy algorithm in less than a second for a representative single site corporate network. We generated thousands of network configurations with the results showing that computing COAs for enterprise networks can be done in a reasonable time period, usually with optimal solutions.

5.2.4 Experiment 3: Full cycle integration

As shown in the workflow Figure 1.1 of Chapter 1, the novel work in this thesis can be applied to computer network defence by assigning rank weights to attack graphs and giving decision support to computer network defence by computing courses of action. Theoretical research forms a critical foundation but application of the work is also a significant research and development effort. This section

presents the integration of the work in this thesis with MulVAL and Trend Micro's Deep Security, a commercial host-based computer security product including a firewall and deep packet inspection intrusion prevention. The project name is Genesis: Integrated end-to-end decision support².

5.2.5 Network architecture

A test network was designed and implemented based upon a complete small business network, with the exception of printers because they were not available. The network architecture is presented in Figure 5.10, and is based on the network presented in the third AssetRank experiment. There are four zones in the network, plus the Internet zone. Traffic between zones is controlled by means of Linux routers, and only traffic deemed necessary for business activities is permitted. While only three workstations are shown, the three workstations could each represent any number of similarly configured hosts. In the environment only the three workstations were implemented but if the environment included twenty-five computers of each workstation type, the vulnerabilities and possible attacks would not change. The configured network zones are:

- DMZ (192.168.100.0/24): DMZ zone for public facing servers. The computers in this zone host HTTP, DNS, and mail services.
- Corporate (192.168.1.0/24): For organization services and end-user workstations. The computers in this zone host mail, source code management, and intranet services, along with workstations.

²The Genesis prototype was funded by Defence Research & Development Canada.

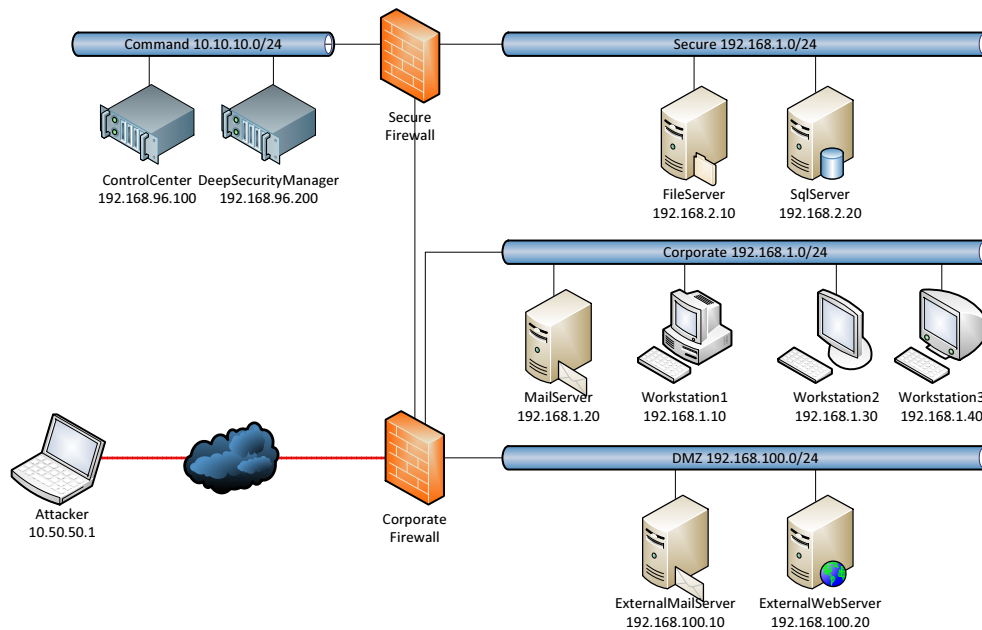


Figure 5.10: Genesis Network: The attacker’s goal is to gain control of “SQLServer” in the “Secure” zone.

- Secure (192.168.2.0/24): For organization infrastructure critical to successfully conducting business. The computers in this zone host database, file, and SSH services.
- Command (192.168.96.0/24): Secured command network for environment security control management. The computers in this zone manage the network security; compute attack graphs, ranking, and course of action recommendations; and implement courses of action in the network.
- Internet: The computers in this zone can access public DMZ services.

All of the servers in the network have been hardened according to NIST security configuration guidelines [74]. Nonetheless, the network is vulnerable to attack

because of the presence of several software vulnerabilities.

5.2.6 Implementation decisions

Determining the allowed host-to-host connectivity is required as input to MulVAL. In order to make this task more manageable, the decision was made not to enforce connectivity restrictions at the Secure Firewall and Corporate Firewall but rather to perform enforcement at each host using the Deep Security Agent. There were several reasons for this decision.

1. An existing tool to gather the firewall rules and parse them for input to MulVAL does not exist and so would need to be built. It is also true that a tool to gather firewall rules from Deep Security Manager must be built but expertise to build that tool was available from Trend Micro.
2. Firewalls at zone borders cannot enforce connectivity restrictions between hosts inside the zone. Connectivity COA recommendations arising from MulVAL attack graphs are granular at the level of preventing access to a specific port between two specific hosts. Intrazone enforcement can only be performed at the host level (in our implementation, by the Deep Security Agent). For interzone enforcement, a defence-in-depth approach was considered where enforcement is performed both on the physical firewall and the host but the purpose of the implementation was to show that the Observe, Orient, Decide, Act (OODA) loop can be fully automated and that purpose can be accomplished more efficiently by removing the complication of multiple-location enforcement.

3. Gathering the actual state of allowed connectivity from both the physical firewalls and the host would have complicated the gathering process.
4. Computing the effective allowed connectivity from the combination of multiple overlapping firewalls with differing policies would have taken a significant implementation effort (for example, Williams *et al.* [88]).

Thus, the Corporate Firewall and Secure Firewall function as routers while the enforcement of the desired connectivity is centrally managed by the Deep Security Manager and is enforced by the Deep Security agents on each host.

5.2.7 Interface and attack scenario

In this scenario, we assume the defender has prioritized the protection of the database server (SQLServer). The server is located behind two routers and it stores the critical data for the business. Due to the network architecture, the attacker cannot reach the database server directly but must first exploit a computer in the DMZ zone. From that point he can reach the computers in the Secure zone.

There are 1,042 deep packet inspection rules that apply to the nine hosts. We have frozen data updates in our experimental network as of March 30, 2010, so that is the date of the newest deep packet inspection rule available. We activated all rules up to March 25, 2010, which corresponds with allowing vulnerabilities that are 5 days old and newer. This leaves 14 deep packet inspection rules available for activation, and makes available the vulnerabilities associated with them to the MulVAL attack graph generator.

The user interface contains five tabs organized in a bread crumb trail style. The work flow follows Figure 1.1, with one tab for each section of the work flow.

Figure 5.11 shows the screen for the data collection phase (Observe). Here, the operator tasks Deep Security to collect information about the network hosts and present the services detected, vulnerabilities, and further information about each vulnerability, when available.

Figure 5.12 shows the screen for the attack graph computation phase (Orient). In this phase, the operator selects the attackers' starting location and the host(s) to prioritize for defence. When the operator clicks the analyze button, the MulVAL input predicate file that contains the network facts is built, and a MulVAL process is started to compute the attack graph.

Figure 5.13 shows the screen for the AssetRank phase (Decide (AR)). The MulVAL attack graph that was output from the previous phase, and the operator's choices regarding the prioritized host(s) to defend are available as input to AssetRank. The operator enters a damping factor for AssetRank and clicks the Analyze button to start an AssetRank process that computes a rank for each attack graph vertex. Also in this phase, a simple heuristic is used to assign costs to each vertex.

In Figure 5.14, the screen for the Course of Action phase is shown (Decide (COA)). The attack graph with rank and cost weights that was output from the previous phase is available for input to the partial-cut vertex set algorithms. The operator selects an algorithm for the computation and enters a budget. The cost of each vertex can also be edited in this screen. When the operator clicks the Analyze button, the courses of action are computed and presented to the user. We see that a budget of 3 was selected which resulted in a course-of-action that eliminates 40% of the rank in the attack graph.

Note that a large proportion of the graph rank can be removed by intelligent deletion of a relatively small set of SINK vertices. This illustrates the supermodular

non-linear aspect of the optimization. The attack graph for this example has 314 SINK vertices, yet removal of just three judiciously chosen vertices (the COA set) is sufficient to remove 40% of the rank in the graph, thereby significantly limiting the attackers' range of action. Similarly, the sum of the (non-infinite) costs of all SINK vertices is 1,813, yet 100% of the rank can be eliminated with a cost of just 5, if the SINK removals are optimal.

Figure 5.15 gives the Act phase screen. The computed COAs are presented to the Deep Security Manager which implements the actions on the hosts. Clicking the Next button restarts the process with a new data collection phase.

Command and Control Centre

Observe
Orient
Decide (AR)
Decide (CoA)
Act
⚙

Vulnerability information loaded: 1/10/2011 5:55:57 PM

Back

Vulnerability Collection

Next

Network vulnerability information must be collected from operating computers and services in order for an attack path to be generated. For the purposes of this prototype, the Trend Micro Deep Security Agent has been deployed to all environment computers, and each computer has been scanned for recommended DPI protection based on detected known vulnerabilities which exist on the operating system and applications installed. Once collected, the vulnerability information can be viewed below.

Collect vulnerabilities
Hide collected vulnerabilities: 1/10/2011 5:55:57 PM

Host: ExternalWebServer (192.168.100.20)			
Name:	ExternalWebServer	ID:	1
IP Address:	192.168.100.20		
Application: Mail Server Common			
Name:	Mail Server Common	ID:	
Ports:	25,110,143		
Filter: SpamAssassin Milter Plugin 'mfi_envrcpt0' Remote Arbitrary Command Injection Vulnerability			
Name:	SpamAssassin Milter Plugin 'mfi_envrcpt()' Remote Arbitrary Command Inject...		
TBUID:	585F1F72-78BA-E5C7-75AA-83711A9306B7		
Severity:	4	CVSS:	-1.00
Application: Web Client Common			
Name:	Web Client Common	ID:	
Ports:	8080,80		
Filter: cURL/libcURL HTTP 'Location:' Redirect Security Bypass Vulnerability			
Name:	cURL/libcURL HTTP 'Location:' Redirect Security Bypass Vulnerability		
TBUID:	63F9E2BC-4483-7220-FFEB-9E3DDA116F9C		
Severity:	2	CVSS:	6.80

[Vulnerability: CVE-2009-0037](#)
[Vulnerability: SA34399](#)
[Vulnerability: SA34259](#)
[Vulnerability: SA34255](#)
[Vulnerability: SA34251](#)
[Vulnerability: SA34237](#)
[Vulnerability: SA34202](#)
[Vulnerability: SA34138](#)
[Vulnerability: 33962](#)

Host: ExternalMailServer (192.168.100.10)			
Name:	ExternalMailServer	ID:	2
IP Address:	192.168.100.10		
Application: Mail Server Common			
Name:	Mail Server Common	ID:	

Figure 5.11: Genesis: Observe phase screenshot

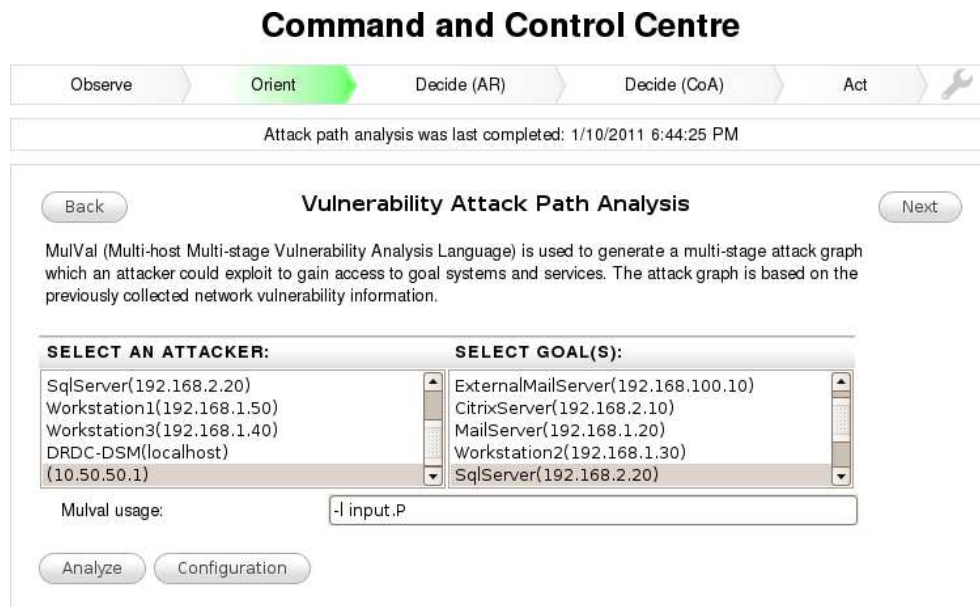


Figure 5.12: Genesis: Orient phase screenshot

Command and Control Centre

Observe Orient Decide (AR) Decide (CoA) Act ⚙️

AssetRank analysis was last completed: 1/10/2011 6:46:00 PM

Back
AssetRank Analysis
Next

AssetRank, is our graph analysis tool which consumes the resulting MulVAL attack graph and generates an understanding of the dependence that attackers have on the vulnerabilities, services, and network routes. It will assign a metric to the attack graph vertices, producing asset ranked results in order of priority which can be used to prioritize courses of action.

Damping factor: Include Visualization?

Analyze Configuration

ASSET RANK ANALYSIS RESULTS

Primary: Secondary: Filter

<< [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) >>

ID	LABEL	TYPE	RANK
71	vulExists('192.168.2.10','1984','WindowsServicesRPCClient',remoteExploit,privEscalator)	SINK	0.01627
67	attackerLocated('10.50.50.1')	SINK	0.01594
90	networkServiceInfo('192.168.2.10','WindowsServicesRPCClient',_139,svcAccount)	SINK	0.00813
92	networkServiceInfo('192.168.2.10','WindowsServicesRPCClient',_445,svcAccount)	SINK	0.00813
75	hacl('192.168.1.30','192.168.2.10',_445)	SINK	0.00092
77	hacl('192.168.1.40','192.168.2.10',_445)	SINK	0.00092
89	hacl('10.50.50.1','192.168.2.10',_445)	SINK	0.00092
73	hacl('192.168.1.20','192.168.2.10',_445)	SINK	0.00092
79	hacl('192.168.1.50','192.168.2.10',_445)	SINK	0.00092
85	hacl('192.168.2.10','192.168.2.10',_445)	SINK	0.00092
87	hacl('192.168.2.20','192.168.2.10',_445)	SINK	0.00092
81	hacl('192.168.100.10','192.168.2.10',_445)	SINK	0.00092
83	hacl('192.168.100.20','192.168.2.10',_445)	SINK	0.00092
50	hacl('192.168.1.20','192.168.2.10',_139)	SINK	0.00092
56	hacl('192.168.1.50','192.168.2.10',_139)	SINK	0.00092
58	hacl('192.168.100.10','192.168.2.10',_139)	SINK	0.00092
52	hacl('192.168.1.30','192.168.2.10',_139)	SINK	0.00092
54	hacl('192.168.1.40','192.168.2.10',_139)	SINK	0.00092
60	hacl('192.168.100.20','192.168.2.10',_139)	SINK	0.00092
66	hacl('10.50.50.1','192.168.2.10',_139)	SINK	0.00092
64	hacl('192.168.2.20','192.168.2.10',_139)	SINK	0.00092
62	hacl('192.168.2.10','192.168.2.10',_139)	SINK	0.00092
49	hacl('192.168.2.10','192.168.100.20',_110)	SINK	0.00037
47	hacl('192.168.100.20','192.168.100.20',_110)	SINK	0.00037
45	hacl('192.168.100.10','192.168.100.20',_110)	SINK	0.00037

Figure 5.13: Genesis: Decide (AssetRank) phase screenshot

Command and Control Centre

Observe Orient Decide (AR) Decide (CoA) Act ⚙️

Course of Action calculation was last completed: 2011-01-10 18:52:23.418979

Back Course of Action Next

The course-of-action engine generates a course of action, taking into account the removal costs and the budget. The course of action suggests patches to apply, services to shut down, and network routes to cut, to maximally disrupt an attacker's ability to control target computers.

Budget Amount: Calculate Algorithm:

Calculate Configuration

COURSE OF ACTION ANALYSIS RESULTS

Primary: Secondary: Filter

<< 1 2 3 4 5 6 >>

ID	LABEL	TYPE	RANK	COST	GROUP	COA
71	vulExists('192.168.2.10','1984','WindowsServicesRPCClient',remoteESINK		0.01627	1	3	
378	vulExists('192.168.1.40','1984','WindowsServicesRPCClient',remoteESINK		0.01627	1	2	
522	vulExists('192.168.1.20','1984','WindowsServicesRPCClient',remoteESINK		0.01627	1	1	
419	hacl('192.168.1.20','192.168.1.30',_143)	SINK	0.00037	5		
423	hacl('192.168.1.40','192.168.1.30',_143)	SINK	0.00037	5		
398	netAccess('192.168.1.40',_445)	OR	0.00813	0		
396	hacl('10.50.50.1','192.168.1.40',_445)	SINK	0.00092	5		
409	hacl('192.168.2.10','192.168.1.30',_110)	SINK	0.00037	5		
399	networkServiceInfo('192.168.1.40','WindowsServicesRPCClient',_44'SINK		0.00813	20		
431	hacl('192.168.2.10','192.168.1.30',_143)	SINK	0.00037	5		
465	hacl('192.168.1.40','192.168.1.30',_80)	SINK	0.00037	5		
461	hacl('192.168.1.20','192.168.1.30',_80)	SINK	0.00037	5		
483	hacl('192.168.1.20','192.168.1.30',_8080)	SINK	0.00037	5		
473	hacl('192.168.2.10','192.168.1.30',_80)	SINK	0.00037	5		
440	hacl('192.168.1.20','192.168.1.30',_25)	SINK	0.00037	5		
487	hacl('192.168.1.40','192.168.1.30',_8080)	SINK	0.00037	5		
452	hacl('192.168.2.10','192.168.1.30',_25)	SINK	0.00037	5		
444	hacl('192.168.1.40','192.168.1.30',_25)	SINK	0.00037	5		
394	hacl('192.168.2.20','192.168.1.40',_445)	SINK	0.00092	5		
370	hacl('192.168.2.10','192.168.1.40',_139)	SINK	0.00092	5		
368	hacl('192.168.100.20','192.168.1.40',_139)	SINK	0.00092	5		
374	hacl('10.50.50.1','192.168.1.40',_139)	SINK	0.00092	5		
372	hacl('192.168.2.20','192.168.1.40',_139)	SINK	0.00092	5		
347	hacl('192.168.1.40','192.168.1.50',_8080)	SINK	0.00046	5		
343	hacl('192.168.1.20','192.168.1.50',_8080)	SINK	0.00046	5		

Total Rank Eliminated: 0.40380
Total Vertices Remaining: 376
Total Arcs Remaining: 528
Total Generation Time: 1.86

Figure 5.14: Genesis: Decide (COA) phase screenshot

Command and Control Centre

Observe Orient Decide (AR) Decide (CoA) Act 

Back **Implement Action** Next

The final step of the integration is to implement determined course of action within the environment. Once the any environment variables are changed, the process can be repeated to re-assess the newly orientated security posture.

Implement Configuration

GROUP	ACTION		
3	vulExists('192.168.2.10','1984','WindowsServicesRPCClient',remoteExploit,privEscalation)	<input checked="" type="checkbox"/>	
2	vulExists('192.168.1.40','1984','WindowsServicesRPCClient',remoteExploit,privEscalation)	<input checked="" type="checkbox"/>	
1	vulExists('192.168.1.20','1984','WindowsServicesRPCClient',remoteExploit,privEscalation)	<input checked="" type="checkbox"/>	

Figure 5.15: Genesis: Act phase screenshot

5.2.8 Trimmed scenario

For our second demonstration we trim the network to only contain the two DMZ-zone computers and the SQLServer host so that the attack graph is small enough to allow a comparison between our polynomial and exponential time algorithms. No deep packet inspection rules are initially activated. Following is some information for this scenario.

Vulnerabilities on ExternalMailServer: 6

Vulnerabilities on ExternalWebServer: 6

Vulnerabilities on SQLServer: 10

Number of vertices in attack graph: 294

Number of arcs in attack graph: 424

Number of individual configuration changes (graph SINKS): 135

In this experiment we used the BFS-PCVS + BruteForcePCVS hybrid algorithm discussed in Section 4.4.7 and obtained interesting results. Table 5.8 gives a listing of the time, total rank, and cost computed by each algorithm.

Budget amounts from 1 to 13 were used and, in all but one case, the entire budget was spent. We stopped computing with the BruteForcePCVS algorithm after a full solution was found. With budgets of 1 to 5, all three algorithms produced solutions with the same total rank. However, with a budget of 7 to 13, the BruteForcePCVS algorithm could find a perfect solution while the BestFirstSearch algorithm continued finding partial solutions. The purpose of the Hybrid algorithm is not to find a solution with a higher rank, but only to find a less expensive solution

Budget	Best-first search			Brute-force			Hybrid		
	Time (s)	Rank	Cost	Time (s)	Rank	Cost	Time (s)	Rank	Cost
1	0.5	0.10	1	0.5	0.10	1	0.5	0.10	1
2	0.9	0.17	2	5.6	0.17	2	1.0	0.17	2
3	1.3	0.24	3	38.9	0.24	3	1.4	0.24	3
4	1.6	0.29	4	207.2	0.29	4	1.9	0.29	4
5	2.8	0.33	5	836.6	0.33	5	3.5	0.33	5
6	3.7	0.37	6	2693.5	0.50	6	5.1	0.37	6
7	4.5	0.40	7	7409.5	1.00	7	7.4	0.40	7
8	5.0	0.43	8				11.0	0.43	8
9	5.4	0.45	9				17.5	0.45	9
10	5.7	0.47	10				30.0	0.47	10
11	5.9	0.49	11				54.8	0.49	11
12	6.1	0.70	12				105.2	0.70	12
13	6.2	1.00	13				144.1	1.00	7

Table 5.8: Algorithm time and accuracy comparison. Ranks and costs that differ for the same budget are indicated in bold.

by identifying redundant COA items that are already removed as a result of removing the rest of the COA items. The Hybrid algorithm only identified improvements in the BestFirstSearch solution with a total rank of 1. In that case, it computed a solution with a cost of 7 — the same cost as the least-cost perfect solution found by the BruteForcePCVS algorithm. However, the time to find the solution was 2.4 minutes compared with 123.5 minutes. This example demonstrates the potential of the hybrid approach in balancing computation time with solution quality. The solution found that completely removes the attack to SQLServer consists of patching 6 vulnerabilities on ExternalMailServer and 1 vulnerability on SQLServer. The patches in this proof of concept consist of activating deep packet inspection rules that block attacks. On the mail server, the activated rules cover the mail software, samba software, and Firefox web client. On SQLServer, the activated rule was for a remotely exploitable vulnerability in the MySQL database software.

Chapter 6

Conclusion

6.1 Summary

In this thesis we reviewed the four main categories of graphs (undirected, directed, hypergraphs, and directed hypergraphs), their matrix representations, and their analysis. Our attention focussed on forward hypergraphs (F-graphs), a kind of directed hypergraph. F-graphs correspond with AND/OR directed graphs and are used to represent the conjunctive and disjunctive dependencies that are present in many real-world applications. F-graphs and AND/OR directed graphs are able to encode the same logic statements; the difference between the two graph types is that F-graphs capture logic semantics via compound arcs while AND/OR graphs do so via typed vertices. We created Algorithms 3.1 and 3.2 to provide a means of converting between the two graph types.

The first goal of the thesis was to provide a method to measure the extent to which nodes support their network (modelled by an F-graph) in the context of its priorities. To that end, we created the AssetRank algorithm, which ranks ver-

tices of AND/OR graphs, and provides vertex-specific damping. We gave a proof of solution existence, applied the algorithm to computer network defence, and integrated vulnerability reference information. We showed how the parameters of our algorithm provide a means for users to model the facets of many networks. Namely:

Personalization vector: Provides a means to specify top-level priorities;

Arc weights: Provide a means to specify local dependency preferences when disjunctive options exist; and

Vertex damping: Provides a means to specify local dependence on factors not included in the graph.

The second goal of the thesis was to provide methods to optimize the separation of source and target subcommunities in networks (modelled by F-graphs) while considering:

1. Node removal costs that are flexible enough to account for impacts to the network, human resources and capital resources;
2. Constrained resources (removal budget);
3. Cascading removals resulting from the removal of a node;
4. Multiple removals combined into a single conjunctive removal; and
5. Overall sum of the ranks for conjunctive node removals and all resulting cascades.

To that end, we defined the concept of a vertex cut-set closure and a closure-relation graph. We created and implemented a linear-time algorithm to compute cut-set closures and a polynomial-time algorithm to approximate the optimal solution since computing the optimal solution in an NP-hard problem. The algorithms take as input an AND/OR directed graph and address the optimization goals given in the previous list through:

1. Vertex cut-set loss costs;
2. Budget constraints;
3. Vertex closures;
4. Partial-cut vertex sets; and
5. Rank sums of cut-set closure vertices.

Computer-network-defence attack graphs provide a ready means for generating AND/OR graphs, which encode the possible ways an attacker can execute a multi-hop attack in a computer network. Networks defenders can use F-graphs and our algorithms as the following sequence shows:

F-graph application: Encode interdependencies between business operations and the network infrastructure, and interdependencies within the infrastructure in an operation-dependency graph;

AssetRank application: Determine the services and hosts that are the most important to the functioning of the system by providing the operation-dependency graph and the operations' relative priorities as input to AssetRank;

F-graph application: Use available tools to produce an attack graph where the goals to defend (defence goals) are the priority services and hosts identified by AssetRank;

AssetRank application: Determine the relative dependence that attackers place on the attack enablers (attack assets) by providing the attack graph and defence goals as input to AssetRank, resulting in a ranked attack graph;

F-graph application: Assign loss costs to the attack graph sink vertices that indicate the operational impact, financial cost, and/or person-hours cost of removing the vertex; and

Partial-cut application: Optimize conjunctive courses of action with the goal of maximally disrupting attackers' abilities to attack the defence goals by providing the attack graph (including ranks and costs) and budget as input to our best-first search partial-cut algorithm, resulting in a conjunctive cut-set providing a course-of-action.

Similarly, other networks can use the work in this thesis to compute courses-of-action. We give as examples, social influence networks, biological contagion networks, and malware contagion networks.

Social influence networks: An F-graph can encode cascading influence between objects (*e.g.*, people, media, institutions). Disrupting connectivity between communities corresponds with weakening the influence that a set of institutions have on a set of people. The ranks give the degree to which each object contributes to influence propagation. Removing a vertex corresponds with instilling distrust so that the influence relationship is broken.

Biological contagion network: An F-graph can encode possibilities for contagion spread between people and would include objects such as light switches, door handles, and ventilation systems. Disrupting connectivity between the source and target communities corresponds with preventing an infected population from spreading disease to a healthy population. The ranks give the degree to which each person and object enables the spread of the contagion. Removing a vertex corresponds with inoculating key people, targetting cleaning of acute contagion-spreading objects, and targetting filtration improvements.

Malware contagion network: An F-graph can encode vectors for malware propagation between computers and would include host configurations, network and air-gapped connectivity, and user-awareness. Similar to the biological contagion network, disrupting connectivity between communities corresponds with preventing infected hosts from infecting healthy hosts. The ranks give the degree to which the host configurations, connectivity, and users enable the spread of the malware. Removing a vertex corresponds with inoculating hosts via updated security tools, targetted connectivity reduction, and targetted user training.

Previous work advanced the state-of-the-art in computer network defence by producing attack graphs. However, attack graphs are too complex to give actionable information to network defenders. This thesis advanced the art by providing the means to rank operational-dependency graphs, rank attack graphs, and produce conjunctive courses-of-action that provide network defenders with optimized actionable information that takes into account the priority of operations

and constrained resources.

6.2 Limitations and suggested work

The work in this thesis can analyze a snapshot of a network, and can consider changing operational priorities, maturation of exploits targetting vulnerabilities, and various other changes through algorithm parameters. However, networks change over time and this work does not address aspects related to rates of change over time (momentum). For example, while AssetRank can provide ranks for different personalization values, it does not give an indication of the rate at which ranks will change as personalization values change.

Another example of time-related limitations can be given by a use case. Suppose an operation is critical for a period of one week, after which time the operation will cease (*e.g.*, a network supplying race results at an Olympic game). Say that a vulnerability is discovered on the fourth day that makes the network extremely vulnerable to attack but fixing the vulnerability requires three days for testing and patch roll-out. Our partial-cut approach to course-of-action decision-support could recommend deploying resources to patch the vulnerability even though the network will be shut down by the time the work has completed. The ability to produce a continuous optimization of network security over a time period would be useful in practice.

We could not validate the correctness of AssetRank with certainty because there is no accepted “right answer” as to what the rank of graph vertices should be. The well-known PageRank algorithm suffers the same problem where it is not clear how to set the algorithm parameters nor what a “correct” ranking is. To test

AssetRank, we formed hypotheses in very simple networks where observers readily agree that the hypotheses are reasonable. After running the experiments, observers see that AssetRank produces results that align with the hypotheses. We experimented with networks of increasing size and complexity until we reached the point where larger networks would be too complex to produce a tenable hypothesis. Our method produces repeatable results; however, it does not give the satisfaction of measuring against a known correct solution.

Remaining hard problems and suggested work

In the following list we identify remaining problems and suggest developmental work.

Loss cost determination: Requiring users to provide all loss costs in an operational course-of-action decision-support system is not practical. We suggest that the loss costs be initially set by the system, and then refined through interaction with the user as the system is used. The system should learn from the users' selections and adjust loss costs appropriately in order to produce better suggestions. The initial loss cost algorithm and the learning algorithm are suggested as future work.

Non-linear loss costs: In a related vein, loss costs that combine supermodularly would more accurately model real-world situations. For example, it may be the case that two different services can each be taken off-line without a serious impact to operations; however, should both of them be taken off-line at the same time, there could be serious consequences. This situation is not captured with linear loss costs.

Vector loss costs: Our work considers a numeric loss cost that combines linearly but it would be useful to optimize loss costs and budgets given as vectors. For example, an action could have an impact cost, financial cost (*e.g.*, to purchase a component), and a time cost that gives the number of hours to complete the work. Each of these costs should be optimized with a separate budget for each.

Dynamic costs: A final area relating to costs is that of costs that change over time. For example, it may cause only a small impact to bring a service or network route down for two hours but a large impact to bring the same asset down for two days.

Rank momentums: Computing rates of change for ranks in relation to changing graph topology, personalization values, and other AssetRank parameters.

AssetRank parameter effects: Similar to other data mining algorithms, rigorous mathematical study will help to understand the effect of parameters on the rank values, and provide guidance in setting parameter values.

General directed hypergraphs: We have shown how to compute partial cuts in F-graphs because they correspond to logic formulas but we have not considered partial cuts in general directed hypergraphs.

Partial-cut evaluation functions: We considered Return On Investment (ROI) as an evaluation function in our partial-cut algorithms. It would be informative to compare ROI with other evaluation functions.

Search functions: We used the best-first search method to produce a polynomial-time algorithm that estimates a solution for the NP-hard optimization problem. Other search functions could be considered and compared.

Network connectivity model: We used preferential attachment to generate simulated networks based on directed connectivity dependencies. Research is needed to determine if network connectivity follows a power-law distribution, and if so, whether it is well-modelled by a preferential attachment generation algorithm.

Parallelization: The partial-cut algorithms are naturally suited to parallelization; this area is under current investigation.

Directed hypergraph drawing: We do not know of any tools that are capable of ingesting a directed hypergraph and producing a visualization.

Bibliography

- [1] Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. *Proceedings of the 23rd international conference on machine learning*, pages 17–24, 2006.
- [2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.
- [3] H.P Alesso, Peter G Prassinis, and Craig F Smith. Beyond fault trees to fault graphs. *Reliability Engineering*, 12(2):79–92, Jan 1985.
- [4] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, Nov 2002.
- [5] Alper Atamtürk and Vishnu Narayanan. The submodular knapsack polytope. *Discrete Optimization*, pages 333–344, Jan 2009.
- [6] Giorgio Ausiello, Paolo G Franciosa, and Daniele Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. *Italian Conference on Theoretical Computer Science*, LNCS 2202:312–328, 2001.

-
- [7] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs: theory, algorithms and applications*. Springer, 2001.
- [8] Albert-László Barabási. Taming complexity. *Nature Physics*, 1(2):68–70, Nov 2005.
- [9] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, Jan 1999.
- [10] Monica Bianchini, Marco Gori, and Franco Scarselli. Inside PageRank. *ACM Trans. Inter. Tech.*, 5(1):92–128, Feb 2005.
- [11] Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. Directed scale-free graphs. *SODA '03 Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, pages 132–139, Jan 2003.
- [12] Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.
- [13] Phillip Bonacich. Power and centrality: A family of measures. *AJS*, pages 1170–1182, Jan 1987.
- [14] Stephen P Borgatti. Identifying sets of key players in a social network. *Computational & Mathematical Organization Theory*, pages 21–34, Jan 2006.
- [15] Allan Borodin, Gareth O Roberts, Jeffrey S Rosenthal, and Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. *Proceedings*, pages 415–429, 2001.
- [16] Ronald S Burt. *Toward a structural theory of action: Network models of social structure, perception, and action*. Academic Press New York, 1982.

-
- [17] Fan R K Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.
- [18] James S Coleman. *The mathematics of collective action*. Transaction Pub, 2006.
- [19] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrel Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. *Proceedings of the 14th ACM conference on computer and communications security*, pages 204–213, 2007.
- [20] Chris Ding, Xiaofeng He, Parry Husbands, Hongyuan Zha, and Horst D Simon. PageRank, HITS and a unified framework for link analysis. *Proceedings*, pages 353–354, 2002.
- [21] John C Doyle, David Alderson, Lun Li, Steven Low, Matthew Roughan, Stanislav Shalunov, Reiko Tanaka, and Walter Willinger. The “robust yet fragile” nature of the internet. *Proceedings of the National Academy of Sciences*, 102(41):14497–14502, Jan 2005.
- [22] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, and Gordon Woodhull. Graphviz—open source graph drawing tools. *Graph Drawing*, pages 483–484, Jan 2002.
- [23] Ayman Farahat, Thomas LoFaro, Joel C Miller, Gregory Rae, and Lesley A Ward. Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. *SIAM Journal on Scientific Computing*, 27(4):1181–1201, 2006.

-
- [24] Linton C Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, pages 215–239, Jan 1979.
- [25] Noah E Friedkin. Theoretical foundations for centrality measures. *American Journal of Sociology*, pages 1478–1504, Jan 1991.
- [26] Giorgio Gallo, Claudio Gentile, Daniele Pretolani, and Gabriella Rago. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, 80(2):213–237, 1998.
- [27] Giorgio Gallo, Peter L Hammer, and Bruno Simeone. Quadratic knapsack problems. *Mathematical Programming*, 12:132–149, 1980.
- [28] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, 1993.
- [29] Giorgio Gallo and Bruno Simeone. On the supermodular knapsack problem. *Mathematical Programming*, 45:295–309, Jan 1988.
- [30] Stephen Guattery and Gary L Miller. On the quality of spectral separators. *SIAM Journal of Matrix Anal. Appl.*, 19(3):701–719, 1998.
- [31] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. Exploring network structure, dynamics, and function using networkx. *Proceedings of the 7th Python in Science Conference (SciPy)*, pages 11–15, 2008.
- [32] Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 11(9):1074–1085, 1992.

-
- [33] Taher H Haveliwala and Sepandar D Kamvar. The second eigenvalue of the Google matrix. *A Stanford University Technical Report* <http://dbpubs.stanford.edu>, 8090:2003–2020, 2003.
- [34] Michiel Hazewinkel, editor. *Encyclopaedia of Mathematics*. Number ISBN 1-4020-0609-8. Springer-Verlag, New York, 2002. <http://eom.springer.de/>.
- [35] John Homer and Xinming Ou. SAT-solving approaches to context-aware enterprise network security management. *IEEE Journal on selected areas in communications*, 27(3):315–322, 2009.
- [36] John Homer, Xinming Ou, and D Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Kansas State University Technical Report*, pages 1–15, 2009.
- [37] John Homer, Ashok Varikuti, Xinming Ou, and Miles A McQueen. Improving attack graph visualization through data reduction and attack grouping. *Visualization for Computer Security (VizSec)*, LNCS 5210:68–79, 2008.
- [38] Charles H Hubbell. An input-output approach to clique identification. *Sociometry*, 28(4):377–399, 1965.
- [39] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. *22nd Annual Computer Security Applications Conference (ACSAC)*, pages 121–130, Dec 2006.
- [40] Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. *SODA '09 Proceedings of the twentieth An-*

-
- nual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237, Jan 2009.
- [41] Sushil Jajodia, Steven Noel, and Brian O’Berry. Topological analysis of network attack vulnerability. *Managing Cyber Threats: Issues, Approaches and Challenges*, pages 248–266, 2005.
- [42] Somesh Jha, Oleg Sheyner, and Jeannette M Wing. Two formal analyses of attack graphs. *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 49–63, Jun 2002.
- [43] Sepandar D Kamvar and Taher H Haveliwala. The condition number of the PageRank problem. *A Stanford University Technical Report*, pages 1–4, Jun 2003.
- [44] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Jan 1953.
- [45] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer Verlag, 2004.
- [46] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *J.ACM*, 46(5):604–632, 1999.
- [47] Bernard Korte and Rainer Schrader. On the existence of fast approximation schemes. *Proceedings of the fourth Nonlinear Programming Symposium*, page 415, 1981.
- [48] Amy N Langville and Carl D Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Review*, 47(1):135–161, 2005.

-
- [49] John Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid and knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, Jan 2010.
- [50] Ronny Lempel and Shlomo Moran. Salsa: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems*, 19(2):131–160, 2001.
- [51] Duan Li, Xiaoling Sun, Jin song Wang, and Ken I M McKinnon. Convergent Lagrangian and domain cut method for nonlinear knapsack problems. *Computational Optimization and Applications*, 42(1):67–104, Jan 2009.
- [52] Lun Li, David Alderson, John C Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2(4):431–523, Jan 2005.
- [53] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, Jan 2007.
- [54] Lung-Tien Liu, Ming-Ter Kuo, Shih-Chen Huang, and Chung-Kuan Cheng. A gradient method on the initial partition of Fiduccia-Mattheyses algorithm. *IEEE International Conference on Computer Aided Design*, pages 229–235, 1995.
- [55] Michael E Locasto, Matthew Burnside, and Darrell Bethea. Pushing boulders uphill: The difficulty of network intrusion recovery. *The 23rd Large Installation System Administration Conference (LISA 2009)*, pages 1–13, Feb 2009.

-
- [56] James B MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
- [57] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette M Wing. Ranking attack graphs. *Proceedings of Recent Advances in Intrusion Detection (RAID)*, LNCS 4219:127–144, Sep 2006.
- [58] Marina Meilă and William Pentney. Clustering by weighted cuts in directed graphs. *Proceedings of the 7th SIAM International Conference on Data Mining*, pages 135–144, Apr 2007.
- [59] Marina Meilă and Jianbo Shi. A random walks view of spectral segmentation. *AI and Statistics (AISTATS)*, pages 1–6, Jan 2001.
- [60] Carl D. Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
- [61] George Nemhauser, Laurence Wolsey, and Marshall Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14:265–294, Jan 1978.
- [62] Steven Noel, Michael Jacobs, Pramod Kalapa, and Sushil Jajodia. Multiple coordinated views for network attack graphs. *IEEE Workshop on Visualization for Computer Security (VizSEC 05)*, pages 99–106, Sep 2005.
- [63] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004*

-
- ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004. ACM Press.
- [64] Steven Noel, Sushil Jajodia, Brian O’Berry, and Michael Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. *19th Annual Computer Security Applications Security Conference (ACSAC)*, pages 86–95, Dec 2003.
- [65] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, 2006.
- [66] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. MulVAL: A logic-based network security analyzer. *14th USENIX Security Symposium*, pages 1–16, Jan 2005.
- [67] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The Page-Rank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.
- [68] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A weakest-adversary security metric for network configuration security analysis. *Proceedings of the 2nd ACM workshop on Quality of Protection*, pages 31–38, Jan 2006.
- [69] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW ’98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM Press, 1998.

-
- [70] Reginald E Sawilla and Craig N Burrell. Course of action recommendations for practical network defence. *Defence R&D Canada Technical Memorandum 2009-130*, pages 1–31, 2009.
- [71] Reginald E Sawilla and Craig N Burrell. Efficient minimum-cost network hardening. *Proceedings of the NATO RTO IST-091 Symposium on Information Assurance and Cyber Defence*, pages 1–22, 2010.
- [72] Reginald E Sawilla and Xinming Ou. Googling attack graphs. *Defence R&D Canada Technical Memorandum 2007-205*, pages 1–24, Jan 2007.
- [73] Reginald E Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. *Proceedings of the 13th European Symposium on Research in Computer Security*, 13:18–34, 2008.
- [74] Karen Scarfone, Wayne Jansen, and Miles Tracy. Guide to general server security: Recommendations of the national institute of standards and technology. *NIST Special Publication 800-123*, Jan 2008.
- [75] Oleg Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon, April 2004.
- [76] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeanette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.

-
- [77] Oleg Sheyner and Jeannette M Wing. Tools for generating and analyzing attack graphs. *Formal Methods for Components and Objects (FMCO 2003)*, LNCS 3188:344–371, Jan 2004.
- [78] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Proceedings*, 22:888–905, 2000.
- [79] David B Skillicorn. Detecting anomalies in graphs. *IEEE Intelligence and Security Informatics*, pages 209–216, 2007.
- [80] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32:41–43, Jan 2004.
- [81] Laura Painton Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. *DARPA Information Survivability Conference & Exposition*, 2:307–321, Jun 2001.
- [82] Steven J Templeton and Karl Levitt. A requires/provides model for computer attacks. *Proceedings of the 2000 workshop on new security paradigms*, pages 31–38, Jan 2001.
- [83] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Aug 2007.
- [84] Dorothea Wagner and Frank Wagner. Between min cut and graph bisection. *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 744–750, 1993.

-
- [85] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, Nov 2006.
- [86] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Toward measuring network security using attack graphs. *QoP '07: Proceedings of the 2007 ACM workshop on Quality of protection*, pages 49–54, Oct 2007.
- [87] Leevar Williams, Richard Lippmann, and Kyle Ingols. An interactive attack graph cascade and reachability display. *Proceedings of the Workshop on Visualization for Computer Security (VizSec)*, pages 221–236, Jan 2007.
- [88] Leevar Williams, Richard Lippmann, and Kyle Ingols. GARNET: A graphical attack graph and reachability network evaluation tool. *Proceedings of the 5th international workshop on Visualization for Computer Security*, pages 44–59, 2008.
- [89] Christoph Witzgall. Mathematical methods of site selection for electronic message systems (EMS). *NASA STI/Recon Technical Report N*, Jan 1975.
- [90] Dengyong Zhou, Jiayuan Huang, and Bernhard Schoelkopf. Learning from labeled and unlabeled data on a directed graph. *Proceedings of the 22nd International Conference on Machine Learning*, pages 1041–1048, 2005.