

SANDIA REPORT

SAND2016-9212

Unlimited Release

Printed September 2016

Final LDRD Report: Using Linkography of Cyber Attack Patterns to Inform Honeytoken Placement

John C. Jarocki, Robert R. Mitchell III, Andrew N. Fisher

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



Final LDRD Report: Using Linkography of Cyber Attack Patterns to Inform Honeytoken Placement

John C. Jarocki
Cyber Enterprise Security Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0799
jcjaroc@sandia.gov

Robert R. Mitchell III
Cyber Security Technologies Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0813
rrmitch@sandia.gov

Andrew N. Fisher
Embedded Systems Analysis Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1027
anfishe@sandia.gov

Abstract

The war to establish cyber supremacy continues, and the literature is crowded with strictly technical cyber security measures. We present the results of a three year LDRD project using Linkography, a methodology new to the field of cyber security, we establish the foundation necessary to track and profile the microbehavior of humans attacking cyber systems. We also propose ways to leverage this understanding to influence and deceive these attackers. We studied the science of linkography, applied it to the cyber security domain, implemented a software package to manage linkographs, generated the preprocessing blocks necessary to ingest raw data, produced machine learning models, created ontology refinement algorithms and prototyped a web application for researchers and practitioners to apply linkography. Machine learning produced some of our key results: We trained and validated multinomial classifiers with a real world data set and predicted the attacker's next category of action with 86 to 98% accuracy; dimension reduction techniques indicated that the linkography-based features were among the most powerful. We also discovered ontology refinement algorithms that advanced the state of the art in linkography in general and cyber security in particular. We conclude that linkography is a viable tool for cyber security; we look forward to expanding our work to other data sources and using our prediction results to enable adversary deception techniques.

Acknowledgements

Thanks to Phil Bennett, Michael Bernard, Jeffrey Bigg, Marshall Daniels, Tyler Dean, David Duggan, Carson Kent, Josh Maine, Marci McBride, Nick Peterson, Katie Rodhouse, Asael Sorenson, Roger Suppona, Scott Watson and David Zage.

We acknowledge support for this work by the LDRD Program at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory operated by Sandia Corporation for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

This page intentionally left blank.

Contents

Preface	13
Summary	15
Nomenclature	16
1 Related Work	19
2 Original Goals	23
2.1 The Linkography Paradigm for Cyber Security	23
2.2 Technology Readiness Level	25
3 Unexpected Threads of Investigation	29
3.1 Behavior Modeling	29
3.2 Linkography Formalization	30
3.3 Subsessionization	30
3.3.1 Truth Data	31
3.3.2 Time Based Substringing	31
3.3.3 Source Based Substringing	32
3.3.4 Linkograph Entropy Based Substringing	32
3.3.5 Statistics Based Substringing	33
3.4 Abstractions	33
3.4.1 Static Cyber Attacker Abstraction	33
3.4.2 Static Cyber Defender Abstraction	34

3.5	Ontologies	34
3.5.1	Self Loop Ontology	34
3.5.2	Forward Progress Ontology	35
3.5.3	Algorithms	35
	Ontology Extraction	36
	Ontology Refinement	36
3.6	Protocol Analysis	37
3.7	Machine Learning	38
3.7.1	Features	38
3.7.2	Behavior Prediction	38
3.7.3	Dimension Reduction	38
3.8	Game Theory	40
3.8.1	Motivation	40
3.8.2	Components	41
3.8.3	Game Type	41
3.8.4	Instrumentation	41
3.8.5	Results	42
3.9	Countermeasures	44
3.9.1	Generating Countermeasures	44
3.9.2	Countermeasure Effectiveness	44
4	Accomplishments	47
4.1	Papers Written	47
4.1.1	Communications and Network Security (CNS) 2015	47
4.1.2	Computing and Communication Workshop and Conference (CCWC) 2017 (Submitted)	47
4.2	Presentations Given	48

4.2.1	CyberSci 2015	48
4.2.2	MITRE Denial and Deception for Cyber Defense Workshop	48
4.2.3	CyberSci 2016 (Accepted)	48
4.3	Artifacts Generated	48
4.3.1	LinkShop	48
	grrcon_example	49
	grrcon_example	49
	hackTestFiles	49
	labels.py	49
	linkoCreate.py	49
	linkoDraw.py	50
	linkoSVG.py	50
	markov.py	50
	runTest.py	50
	stats.py	50
4.3.2	Machine Learning Models	51
4.4	Collaborations Started	51
5	Conclusion	53
5.1	Lessons Learned	53
5.2	Future Work	54
	References	55
	 Appendix	
A	Appendix: Ontology Refinement Algorithms	57

A.1	Brute Force Minimum Similarity Pseudocode	58
A.2	High Impact First Minimum Similarity Pseudocode	59
B	Appendix: Functional Linkography	61
B.1	Motivations	62
B.2	Infrastructure	64
C	Appendix: Deriving Markov Models	67
C.1	Introduction	67
C.2	Representation	67
C.3	Markov Interpretation	69
C.4	Uses	69
C.5	Markov Models of Sessions (Link-Predictor Model)	69
C.6	Markov Models of Sessions (Behavioral Model)	72
C.7	Link-Predictor Markov as a Next State Predictor	73
C.8	Distance Between Matrices	74
D	Appendix: Map Reduce Algorithms	77
D.1	Optimal Ontology Map	78
D.2	Optimal Ontology Reduce	78
D.3	High Impact First Minimum Similarity Reduce	79

List of Figures

2.1	The overall paradigm.	23
2.2	Preprocessing flow detail.	24
2.3	Intrusion detection flow detail.	24
2.4	Create linkograph flow detail.	24
2.5	Behavior prediction flow detail.	25
2.6	Ontology refinement flow detail.	25
2.7	Overall paradigm readiness.	25
2.8	Preprocessing flow readiness.	26
2.9	Intrusion detection flow readiness.	26
2.10	Create linkograph flow readiness.	26
2.11	Behavior prediction flow readiness.	27
2.12	Ontology refinement flow readiness.	27
3.1	Self loop ontology.	34
3.2	Forward progress ontology.	35
3.3	Behavior prediction results.	39
3.4	Forward selection results.	39
3.5	Backward selection results.	40
3.6	Cyber shopping game.	43

List of Tables

3.1	Time based substringing results.	32
3.2	Source based substringing results.	32
3.3	Linkograph entropy based substringing results.....	33
3.4	Statistics based substringing results.	33

Preface

In 2011, University College London Bartlett School of Architecture Professor Alan Penn gave a lunch hour lecture titled “Who enjoys shopping in IKEA?” [21] This video came to our attention just as we were thinking about the problem of tracking progress of cyber attackers through a target environment. The lecture introduced us to the idea of “space syntax” of geographic spaces such as city streets or stores such as Ikea. With navigation of a store as an analogy for the progress of an attacker, we realized we could ask questions like:

1. What part of the attack space can they see from their current vantage point?
2. Can we tell what they are “shopping” for or cause them to reveal this information?
3. Can we distract them, delay them, or convince them to “purchase” something else?
4. Can we control the intelligibility of the attack space in order to control the effectiveness of the attack?
5. Finally, can we understand them at the microbehavioral level by analyzing the relatedness of moves made?

As we were considering potential research topics that would allow us to examine these areas and expose behavioral cues and responses during a sequence of cyber attack steps, we came across the 2012 paper “Order, Structure and Disorder in Space Syntax and Linkography: Intelligibility, Entropy, and Complexity Measures.” [4] This paper introduced us to a discipline called linkography that was released to the protocol analysis community by Gabriella Goldschmidt in 1990. It also gave us the idea that a time series of cyber events (traversing the attack space) could be represented as a series of moves (nodes) in a linkograph. El-Khouly and others [12] had already experimented with using information theory, entropy, and other metrics to perform qualitative and quantitative analysis of linkographs in the space syntax and design creativity fields.

With this information in hand, we had the basic materials to begin our research: We had a protocol analysis methodology not previously used in the cyber domain (linkography), we had a general model of a cyber attack as a series of moves through an attack space, and, we had the basic metrics and methods for extracting quantitative data from data organized in linkographs.

We had a goal at the outset to create a system that would do four things: First, the system would recognize the shape and scope of an attack in progress. Second, it would help us understand the objectives, capabilities, and patterns of specific attackers. Third, the system would identify critical moves during an active attack. Finally, it would place cues (we later coined the term honeycue) to influence attacker behavior in real time.

While we have made good progress toward these goals, we discovered gaps both in the tools and existing modeling techniques; this required our team to spend significant time and effort to overcome. Rather than feeling discouraged by this extra work, we feel we have materially advanced the state of the art for both linkography in general and its application to cyber security specifically.

Some of the areas we addressed include:

1. Formalization of linkography through new definitions and proofs,
2. Characterization of cyber attack steps at a consistent level of abstraction,
3. Deep exploration of entropy as a key measurement of creativity in linkographs,
4. Expansion of the concept of ontology in linkography to create a behavioral models,
5. Automated methods for refining an existing model to match captured activity,
6. Tools to enable human analysts to make adjustments to the refinement process.

The rest of this report will tell the story of our journey to create a scientific, repeatable, and quantitative method for tracking microbehavioral activity of cyber attackers that still keeps the human analyst in the loop. We have attempted to create a reusable framework that can be applied in the context of network, web, database, operating system command, or system call level activity.

Summary

This report provides a survey of linkography-related literature, the original goals of the lab directed research and development (LDRD), unanticipated research directions that spontaneously occurred, a summary of accomplishments and proposals for future work.

Nomenclature

abstraction a set of categories for classifying raw data

backlink a relationship between a given linkograph node and an earlier one

classification a machine learning formulation that maps a data point onto a known category

cleanup one of the five abstraction classes we consider in this work

clustering a machine learning formulation that maps a data set onto a number of unknown categories

critical move a node in a linkograph that has a large number of links

derived linkograph a graph generated from a labeled data set and an ontology

dimension reduction a machine learning formulation that decreases the number of features

entropy a measure of the information content of an artifact

execute one of the five abstraction classes we consider in this work

feature a predictor, or input, of a machine learning model

forelink relationship between a given linkograph node and a later one

horizonlink the links present a given depth from the nodes or the links spanning a given distance between nodes

link a relationship between two linkograph nodes

linkograph a labeled time series and the set of binary relationships between nodes

look one of the five abstraction classes we consider in this work

move a node in a linkograph; one of the five abstraction classes we consider in this work

nonparametric model a machine learning model that does not assume a certain relationship (e.g., linear or polynomial) between the features and the response

ontology an abstraction and a set of relations between its classes

parametric model a machine learning model that assumes a certain relationship (e.g., linear or polynomial) between the features and the response

protocol analysis collecting a raw data set, organizing it into a dialog transcript and transforming it into an abstract form

response the output of a machine learning model

session a block of time series data that is bracketed on either side by a time gap of one hour or more

substring a contiguous ordered subset

subsequence an ordered subset that need not be contiguous

subsession a block of time series data that comprises a single tactical objective

supervised model a model trained with labeled, or truth, data (e.g., a classifier or regression model)

T-code prefix free variable length code proposed by Mark Titchener

transfer one of the five abstraction classes we consider in this work

unsupervised model a model trained without labeled, or truth, data (e.g., a clustering model)

This page intentionally left blank.

Chapter 1

Related Work

This chapter provides a survey of prior work related to this LDRD.

In “Order, Structure and Disorder in Space Syntax and Linkography: Intelligibility, Entropy, and Complexity Measures,” [4] Tamer El-Khouly and Alan Penn make several points: First, the time component distinguishes linkography from other forms of analysis. Also, linkography uses real relative asymmetry (RRA) for a lot of its analysis, so it was not quite applicable to this LDRD. Third, El-Khouly provides a bit more information on the measurement of information and entropy. Also, the authors explain more why small strings have inaccurate entropy due to the bounds of the logarithmic integral used to calculate information and entropy. Finally, it is important to note that subgraphs in a linkograph might not always intersect as they are part of different patterns of actions/thoughts.

Later El-Khouly and Penn build on their previous work in “On an integrated analytical approach to describe quality design process in light of deterministic information theory” [3]: First, the authors explore deterministic information theory by Titchener (T-code sets). They write, “the challenge of this paper is to understand the transformation of ideas from one state to another can be captured.” Third, El-Khouly computes T-complexity and T-entropy (which are based on T-complexity) for each node in the linkograph as well as subgraphs. Next, the authors note that at each node, you can either look at links going forward or links going backwards. Fifth, they observe that subgraphs should be chunked based on number of nodes or time. The most interesting part is that T-codes are designed to work at local levels, and not global.

In “The designer as a team of one,” [10] Gabriela Goldschmidt, creator of linkography, makes several points: First, the paper defines link index as the proportion of links to moves. Also, a high link index tends to be indicative of high productivity. This tends to imply moves that induce the greatest number of links (also known as critical moves) should be analyzed further. Third, the paper references Miller’s Law (7 ± 2) [18] for the size of subgraphs to examine.

Later, Goldschmidt summarizes linkography’s history in her monograph [11]. The work covers decades of humans performing protocol analyses on audio transcripts, video transcripts and interviews to create labeled time series and, in turn, linkographs.

Jeff Kan and John Gero have made a number of contributions to the literature [9], [12], [13], [14], [15], [16]:

In particular, Gero [8] initially proposed the function-behavior-structure schema in AI Magazine Volume 11 Number 4. This schema would later become the FBS ontology that has been widely adopted in the design research community. The FBS ontology provides the categories of design state spaces (Requirements, Function, Expected Behavior, Structure, Behavior derived from structure) and the design activities that transform the design objects between these state spaces. Our work extends this concept and formalizes it for cyber security by defining an ontology as the set of abstraction classes and the relation between those states. This ontology provides a general model for cyber attack behavior in the same way that the FBS ontology does for design. In both cases, the relations between the ontology states provide the rules for transforming a labeled time series into a linkograph. We use the term abstraction to describe the set of classes or categories into which distinct cyber attack moves can be binned. This abstraction is effectively a taxonomy for the type of cyber attack data being modeled.

Gero, et al. [9] also propose a seven stage model of protocol analysis using linkography:

1. coding scheme definition
2. recording
3. transcription
4. segmentation and coding
5. coding analysis
6. link definition
7. linkograph analysis

However, a gap exists in the literature with respect to the automation of step six, link definition, which makes results hard to reproduce. To date, most linkographs have been produced manually, or at least required a subject matter expert to label nodes according to a static ontology. This is an area where we have significantly advanced the field of linkography; our LinkShop code can create a linkograph automatically given inputs of raw time series data and a nominal ontology. Furthermore, if the ontology does not match expectations, a human analyst can provide feedback to refine until it does.

Also, Pourmohamadi and Gero [22] propose Linkographer as a tool to aid in analyzing linkographs. While this tool generates linkographs and linkograph statistics, it does so based on human-defined links. Linkographer visualizes and extracts information from the linkograph that the human creates, but the human still performs the nebulous and time-consuming task of linking events. The authors address step seven (linkograph analysis) of Gero, et al.'s [9] model. While Linkographer is specific to the FBS ontology, our ontology extraction and refinement work adapts to abstraction classes from any ontology.

Next, Kan and Gero discuss using cluster analysis, specifically the SPSS TwoStep cluster algorithm, to automatically identify clusters in a linkograph in [14] and extract other information from a linkograph in [15]. This addresses step seven (linkograph analysis) of Gero, et al.'s [9] model.

Later, in “Can entropy represent design richness in team designing?” [13] Kan and Gero introduce the use of Shannon entropy. The authors say that linkography is the graphical representation of design processes in terms of the links between design moves. They claim higher entropy indicates richer idea generation. Kan and Gero observe that as the size of the linkograph grows, the possible combinations grows rapidly, and they reason for needing theory-based measures.

Following this, in “Comparing entropy measures of idea links in design protocols,” [12] Kan, et al. build on [13]. This paper uses Shannon’s entropy to compare multiple linkographs, assuming a move in a linkograph is a manifestation of ideas and entropy indicates the idea development potential. This paper explicitly shows their Shannon entropy formula. The authors claim backlinks record a path that led to the generation of a move [10]; backlink entropy measures opportunity for enhancement and response. On the other hand, forelinks bear evidence to a contribution to the production of further moves; forelink entropy measures opportunity for creation and initiation. Finally, horizontal links illustrate bias towards cohesive moves and saturation lower (closer to the nodes) in the linkograph; horizontal link entropy relates local thought to big ideas. Link measures tend to be inflated in fully saturated graphs. This invites the criticism that these linkographs may actually be less novel (and Kan and Gero think so also). Basically, empty or complete graphs are boring: Specifically, an empty or sparse linkograph suggests a non-converging process with no coherent flow. On the other hand, a full linkograph suggests a fully integrated process with no diversity. Kan, et al. view linkograph creation as a 2-stage manual process.

In their subsequent work titled “Acquiring information from linkography in protocol studies of designing,” [15] Kan and Gero looked at using Shannon entropy and clustering to understand linkographs. The authors identified three patterns of moves: First a chunk is a group of moves linked to themselves. Next, a web is a large number of links among a small set of nodes. Finally, a sawtooth is a special sequence of linked moves. They define link index as the number of links over the number of moves. Kan and Gero identify advantages of linkography: First, it is scalable: structure is not tied to one individual or set of actions; linkographs illustrate multiple individuals and events. Also, it is flexible: moves and links can be done at separate times. The authors identify three core linkography tasks: First, segmentation is what we call tokenization. Next, coding is creating links between moves or states. Finally, analysis is interpreting linkographs. They offer an alternative interpretation of forelinks and backlinks: forelinks are initiations and backlinks are responses. Computer-related linkographs have very different linkograph structures. The paper states saturated linkographs with more than seven moves/states are very rare. We think this is the case in the cognitive domain: Working memory and work have a different context in computing.

Finally, Kan and Gero continue their study of linkography in “About: Designing-Analysing Design Meetings.” [16] The authors observe that different coding schemes yield different graph structures for the same design process. Also, they provide additional motivation for a general ontology: “to develop a general coding scheme that yields high quality, uniform results, that maps well to the behavior of designers, produces a deeper understanding of design thinking and activities and can be applied across protocols independently of the domain and the number of participants. The general coding scheme is based on an ontology of the domain of designing and as a consequence is not an ad hoc development specific to a unique protocol but one that can be used uniformly across design protocols independently of the specific design activity being studied and unrelated to the

number of participants in a design team.”

Becattini, et al. [1] propose a framework based on OTSM-TRIZ [17] to automate protocol analysis. OTSM and TRIZ are acronyms formed from transliterations of Russian terms meaning “General Theory of Powerful Thinking” and “Theory of Inventive Problem Solving,” respectively. The authors’ proposal addresses steps one (coding scheme definition) and seven (linkograph analysis) of Gero, et al.’s [9] model. While their study is tailored for protocol analyses of designers, we have constructed a framework usable for any domain.

Fruchter [7] investigates RECALL, which is a tool that automatically indexes the audio and video of a design session. This addresses steps two (recording), three (transcription) and four (segmentation and coding) of Gero, et al.’s [9] model. RECALL is applicable specifically to the design domain, while our investigation is germane to any activity.

Fisher, et al. [5] propose using linkography to characterize and disrupt cyber attackers. The authors identify the labeled time series to linkograph transform (Gero, et al.’s [9] sixth step) as an open question that we pursue in our ontology extraction and refinement work. In particular, they cast attacker console commands into one of five categories: Look, Transfer, Move, Execute or Cleanup. This allows the cyber security researcher to abstract the attacker’s actions as a set of console commands represented in a time series, with the five categories as labels. Fisher, et al. propose using an ontology to convert the labeled time series into a linkograph in a scientific and repeatable fashion. The resulting linkographs relate points in the labeled time series which allows previously unrealized insights to emerge.

The work of Mitchell, et al. [19] was the logical progression of [5]. In it, the authors propose a technique for adjusting an ontology over time. The existing ontology and a human-corrected linkograph form the input for the algorithm.

Chapter 2

Original Goals

This chapter discusses the conceptual model that illustrates our, now clearer, picture of a linkography based cyber security behavior modeling, detection, and response system. Our work on this LDRD spans the full time domain and functional range of cyber security operations. We visualize the complex network of data and logic that fills space with a hierarchy of flow diagrams. Because some of the logic blocks in these figures are more complete than others, we augment the flow diagrams with indications and discussions of each block's maturity.

2.1 The Linkography Paradigm for Cyber Security

Figures 2.1 - 2.6 illustrate the system that describes our concept of LDRD goals with the benefit of three years of hindsight. The rectangles indicate blocks of logic and the parallelograms show data at rest.

Figure 2.1 is the top level flow diagram comprising all inputs, outputs and processing we envision.

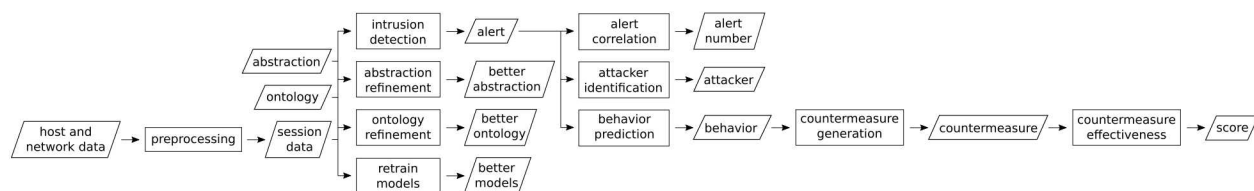


Figure 2.1. The overall paradigm.

Figure 2.2 illustrates the transformation of raw sensor data into something the intrusion detection, abstraction refinement, ontology refinement and retraining flows can accept.

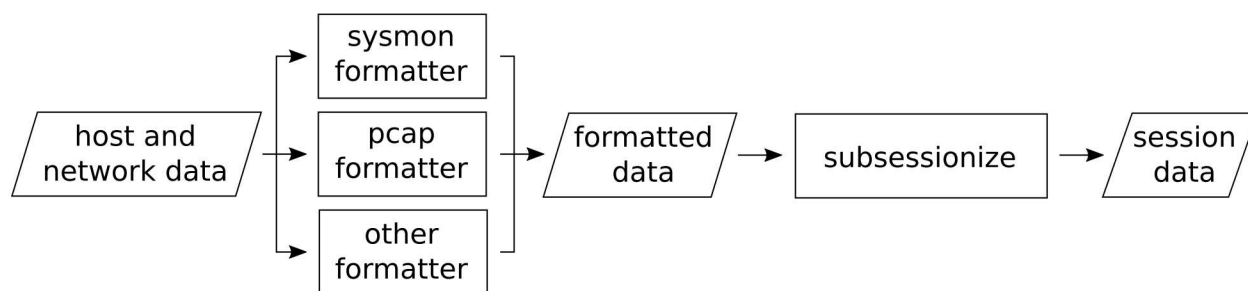


Figure 2.2. Preprocessing flow detail.

Figure 2.3 is very similar to the behavior prediction flow; the main difference is that the apply model flow for the intrusion detection flow will be a binary classifier.

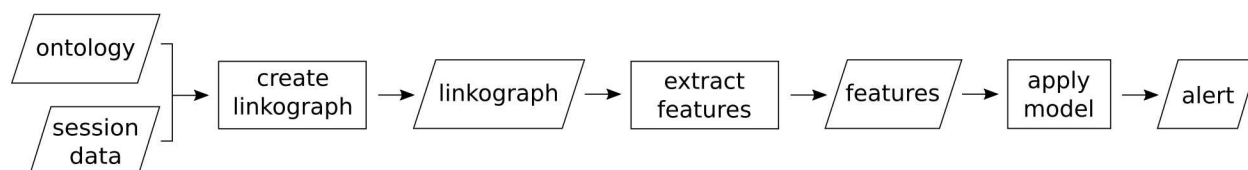


Figure 2.3. Intrusion detection flow detail.

Figure 2.4 is a further decomposition of the create linkograph flow seen in the intrusion detection and behavior prediction flows.

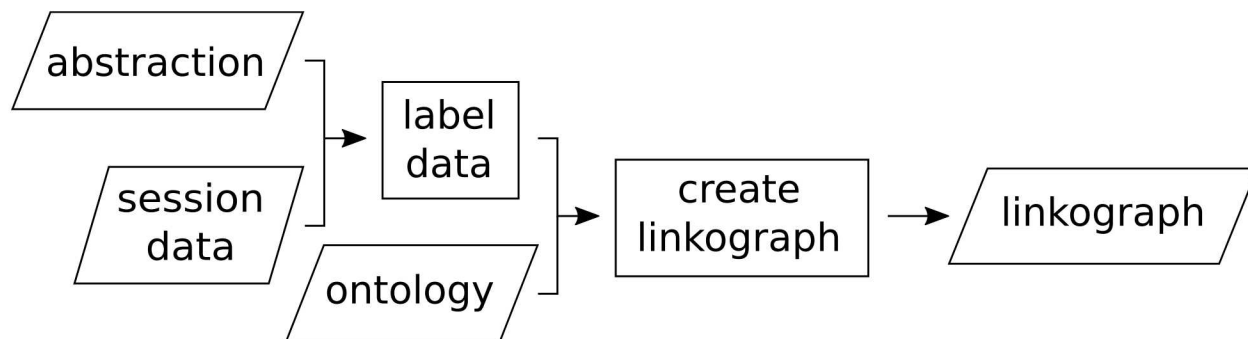


Figure 2.4. Create linkograph flow detail.

Figure 2.5 is very similar to the intrusion detection flow; the main difference is that the apply model flow will be a multinomial classifier for the behavior prediction flow.

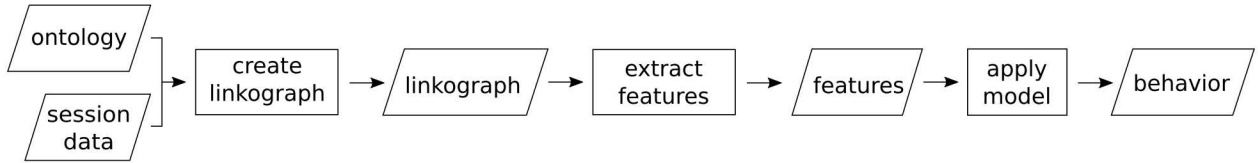


Figure 2.5. Behavior prediction flow detail.

Figure 2.6 illustrates how LDRD artifacts can anneal an ontology.

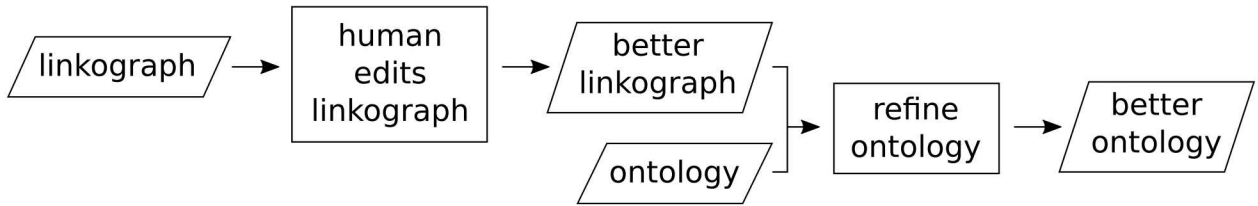


Figure 2.6. Ontology refinement flow detail.

2.2 Technology Readiness Level

The borders of the shapes in Figures 2.7 - 2.12 show our coverage of the LDRD goals and the remaining gaps. Solid-bordered flows are complete, dash-bordered flows are partially complete and dotted flows are opportunities for future work. The state of dash-bordered flows ranges from an interesting thought experiment to a prototype that is almost complete.

The readiness of Figure 2.7 flows is governed by their decompositions:

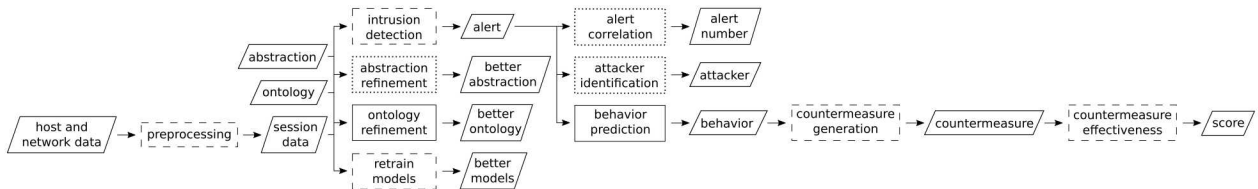


Figure 2.7. Overall paradigm readiness.

In Figure 2.8, `enc.py` is a partial implementation of a flow for ingesting and formatting Microsoft Sysmon data. Technology adopters can implement the other format flows as long as they

produce output in the simple JavaScript Object Notation (JSON) format expected by the LinkShop utilities. The subsessionize flow (`subsessionize.py`) is complete.

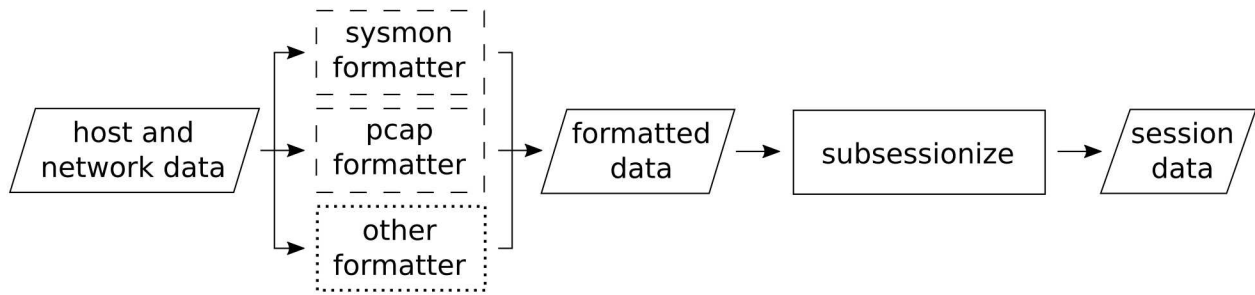


Figure 2.8. Preprocessing flow readiness.

In Figure 2.9, the create linkograph and extract features (`extractSessionFeatures.py`) flows are complete and shared with the behavior prediction flow.

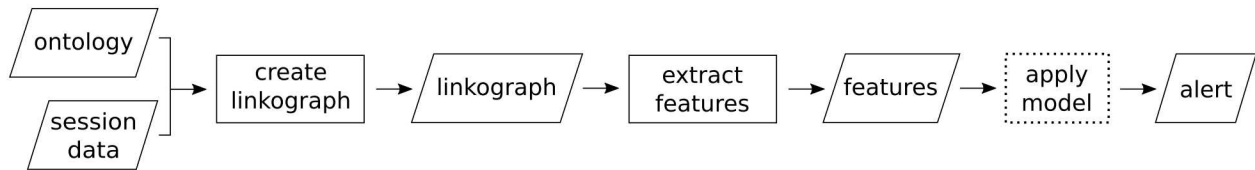


Figure 2.9. Intrusion detection flow readiness.

In Figure 2.10, the labeling (`labels.py`) and linkograph creation (`linkoCreate.py`) flows are complete.

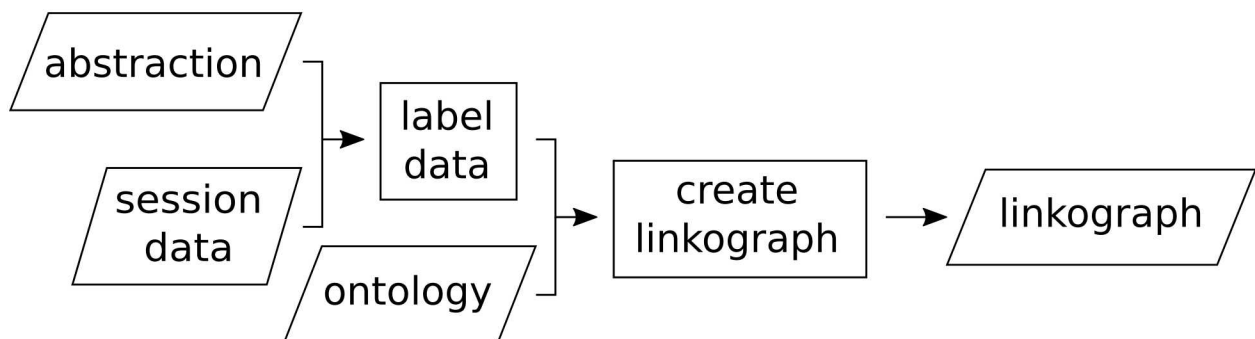


Figure 2.10. Create linkograph flow readiness.

In Figure 2.11, the create linkograph and extract features (`extractSessionFeatures.py`) flows are complete and shared with the intrusion detection flow. We have an R implementation (`multinomial_classification.R`) for the apply model flow; porting this to Python (`scikit-learn`) is straightforward.

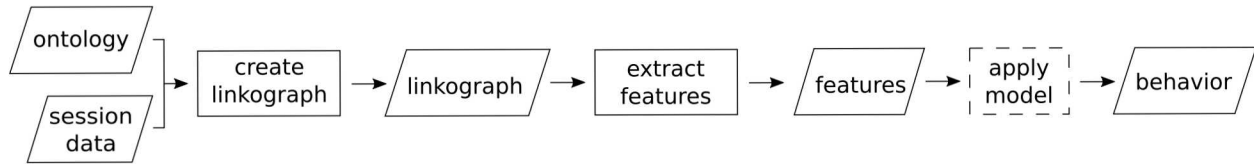


Figure 2.11. Behavior prediction flow readiness.

All flows in Figure 2.12 are complete. Linkshop provides the editing capability, and `ontologyExtraction.py` provides the implementation of the refine ontology flow.

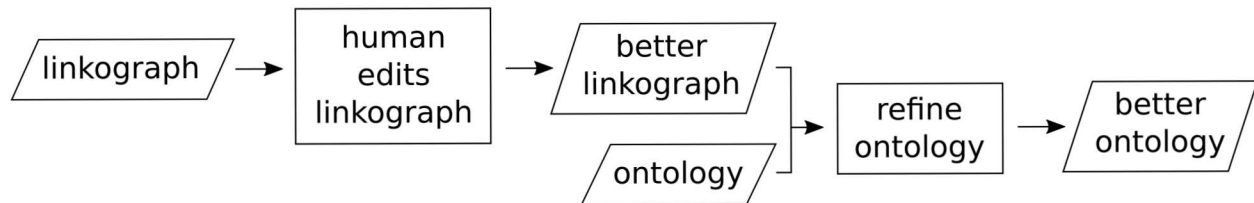


Figure 2.12. Ontology refinement flow readiness.

This page intentionally left blank.

Chapter 3

Unexpected Threads of Investigation

Our topic was completely new, and our work on this LDRD quickly exposed complications that needed to be explored. We needed to do this in order to provide a solid foundation upon which we could produce the results we originally envisioned. This chapter discusses:

1. Behavior Modeling
2. Linkography Formalization
3. Subsessionization
4. Abstractions
5. Ontologies
6. Real-World Protocol Analysis
7. Machine Learning
8. Game Theory
9. Countermeasures

3.1 Behavior Modeling

The team identified a shortage of behavior modeling at the tactical level. Most behavioralists are focused on macrobehavior: the strategies of nation states or other large entities over the course of days to years. In contrast, we are interested in microbehaviors, which are the actions of an individual over the course of seconds to hours. The Linkography methodology we developed closes this gap by providing a way to classify microbehavior steps using an abstraction (essentially a taxonomy) and by relating those steps to each other in an ontology. The ontology becomes a general behavioral model, but by examining specific attack data as linkographs, Markov models can be derived to create a model for a specific actor or even a specific attack session. More details on this process can be found in Appendix C.

3.2 Linkography Formalization

The team initially tried to build on existing linkography tools, but we found a lack of rigor in the prior work. Team members addressed this by proposing the following definitions in the literature:

Formally, a linkograph can be defined as follows:

Definition 1 (Linkograph) A linkograph on N nodes (where $N \in \mathbb{N}$ is a natural number) is a set of ordered pairs, $\mathcal{L}$, such that

$$\mathcal{L} \subset \mathcal{M}_N := \{(i, j) \mid 0 \leq i < j < N\}.$$

The individual natural numbers $0 \leq i < N$ are nodes, and each ordered pair (i, j) , with $i < j$, is a link from node i to node j . In a link (i, j) , node i is the initial node and node j is the terminal node. $\mathcal{M}_N$ is the set of all possible links whose initial node is at least 0 and whose terminal node is less than N . A labeling is a function $L : \mathbb{N}_{<N} \rightarrow \text{Abs}$, where $\mathbb{N}_{<N}$ is the set of natural numbers less than N and Abs is a set of strings called abstraction classes.

Definition 2 (Ontology) An ontology, $\mathcal{O}$, is a pair (Abs, R) where Abs is a set of abstraction class strings and $R \subset \text{Abs} \times \text{Abs}$ is the set of relations between the abstraction classes. The relation $(A, B) \in R$ is written $A \rightarrow B$. A is the initial class and B is the terminal class. If $A = B$, the relation is called a self-loop. The size of an ontology, $|\mathcal{O}|$, is the size of the set of abstraction classes: $|\mathcal{O}| = |\text{Abs}|$.

Definition 3 (Derived Linkograph) Given an ontology, $\mathcal{O}$, and a labeling, $L : \mathbb{N}_{<N} \rightarrow \text{Abs}$, the linkograph, $\mathcal{L}$, defined by

$$(i, j) \in \mathcal{L} \Leftrightarrow L(i) \rightarrow L(j),$$

is the derived linkograph given $\mathcal{O}$ and L and is denoted by $\mathfrak{L}(\mathcal{O}, L)$.

The complete collection of the definitions and proofs contributed by our team can be found in [6].

3.3 Subsessionization

Linkographs of raw cyber attack moves can grow large very quickly. While large graphs can be impressive to look at, our research showed that the statistical significance drops dramatically as the length of a linkograph grows. We wanted to formalize the concept of breaking the linkographs into windows, or sub-linkographs, of activity. This led us to the concept of subsessionization. Sessionizing occurs before subsessionizing: sessionizing is substringing the input stream when 60 minute delays are encountered.

Subsessionizing the input stream (nominally console commands) is a key problem in cyber shopping. The benefits of larger subsession sizes are that they have more context and higher entropy. The drawbacks of larger subsessions are that a human will struggle to make sense of them and complex (e.g., those that grow superlinearly with respect to subsession length) processing algorithms will not perform well. On the other hand, the benefits of smaller subsessions are that humans can infer their goal and high complexity algorithms will be viable. The drawbacks of smaller subsession sizes are that less potentially enriching nodes will be present in these subsessions and the entropy of these subsessions will be lower.

In this activity, we identify the best way to subsessionize the input stream for further processing. We organized subsessionization by two dimensions: The first dimension distinguishes substrings (separating into contiguous strings) from subsequencing (separating into sequences that only preserve the order of the input) techniques while the second dimension separates online from offline strategies. We focused on substrings approaches: Although subsequencing is an interesting approach, substrings the input stream is a step in this direction and should be studied first. We followed this same intuition when pursuing ontology extraction on the way to ontology refinement.

We studied substrings based on time delay, traffic endpoint, linkograph entropy and statistical likelihood. An online strategy was a natural fit for time delay, traffic endpoint and linkograph entropy based techniques. We used an offline strategy for the statistical likelihood based technique.

3.3.1 Truth Data

The data set we used for all evaluations is a sessionized 195 command excerpt from a real world event in `.xlsx` format. Two humans examined session 4, comprising 101 commands, and identified ideal partitionings. We considered these partitionings as our “truth data.”

Reviewer one’s partitioning comprised 7 subsessions of lengths 6, 17, 4, 3, 44, 1 and 26 and durations 22, 5961, 10, 774, 8361, 0 and 1352 s. It has a length mean and variance of 14.4 nodes and 251.0 nodes² and a duration mean and variance of 2354.3 s and 11511462.9 s².

Reviewer two’s partitioning comprised 12 subsessions of lengths 6, 3, 8, 10, 3, 5, 7, 26, 7, 12, 5 and 9 and durations 22, 31, 2576, 807, 774, 42, 262, 2889, 199, 117, 206 and 131 s. It has a length mean and variance of 8.4 nodes and 37.9 nodes² and a duration mean and variance of 671.3 s and 1000809.2 s².

3.3.2 Time Based Substrings

The intuition behind time based substrings is that human actors will break after achieving some tactical objective to evaluate the results and choose their next goal. This algorithm substrings upon encountering a delay that surpasses the maximum delay parameter. Based on reviewer one’s partitioning, the optimal maximum delay is slightly larger than 15 minutes. Based on reviewer

two's partitioning, the optimal maximum delay falls between 7 and 15 minutes. Table 3.1 shows how this approach performed.

Table 3.1. Time based substringing results.

Maximum Delay (s)	Substring Count	Length Mean	Length Variance	Duration Mean (s)	Duration Variance (s ²)
420	15	6.7	24.5	194.7	34808.0
900	8	12.6	225.4	890.6	1307130.8
1800	4	25.2	762.2	3131.2	7483475.6

3.3.3 Source Based Substringing

Source based substringing supposes that a subsession should comprise all contiguous communication from a source address. This is really a strawman approach because it does not withstand scrutiny. First, if two actors are working a victim simultaneously, the source will flap between the two: subsequencing is the answer to this problem (cf. Section 5.2). Second, if a single actor is using a redirector, the subsessions should span address changes to avoid losing coherence. Third, if a single actor not using a redirector compromises a host for the duration of a campaign, their strategic objective will comprise many tactical goals that can be more effectively analyzed separately. This algorithm substrings upon encountering a source address change in the time series. These results match the reviewer one's partitioning statistics exactly. Unfortunately, we cannot count on the availability of source data. Table 3.2 shows how this approach performed.

Table 3.2. Source based substringing results.

Substring Count	Length Mean	Length Variance	Duration Mean (s)	Duration Variance (s ²)
7	14.4	251.0	2354.3	11511462.9

3.3.4 Linkograph Entropy Based Substringing

The intuition behind entropy based substringing is that linkograph entropy will rise when a subsession spans two distinct activities. This algorithm substrings when the linkograph entropy for a subsession surpasses the maximum entropy parameter. These results are interesting: although we can't oversubsessionize (i.e., there is no way to get more than 6 substrings), the results are reasonable over a wide range of maximum entropy (i.e., the interval from 0 to 0.6). Although these results approach reviewer one's partitioning, there is a significant residual separating the optimal entropy based substringing results from reviewer two's partitioning. Table 3.3 shows how this approach performed.

Table 3.3. Linkograph entropy based substringing results.

Maximum Entropy	Substring Count	Length Mean	Length Variance	Duration Mean (s)	Duration Variance (s ²)
0.001	6	16.8	715.0	2328.3	14602237.1
0.5	6	16.8	716.6	2644.8	13453651.4
0.9	3	33.7	1826.3	6711.0	32494332.0

3.3.5 Statistics Based Substringing

The basic idea is to find the largest acceptable subsession inside a session, and then apply the algorithm recursively to the nodes to the left of (earlier in) the subsession and to the nodes to the right (later). In case multiple subsessions of the same size are acceptable, the one with the best fit is chosen. In case all fits have equal quality, the earliest subsession is chosen. For a subsession to be acceptable, it must fulfill three criteria:

1. It must not force a runt subsession to the left or right.
2. The entropy of the subsession linkograph must be within one standard deviation of the mean entropy for subsession linkographs of the same length.
3. The subsession must not contain disconnected linkographs.

Although these results approach reviewer one's partitioning, there is a significant residual separating the statistics based substringing results from reviewer two's partitioning. Table 3.4 shows how this approach performed.

Table 3.4. Statistics based substringing results.

Substring Count	Length Mean	Length Variance	Duration Mean (s)	Duration Variance (s ²)
7	14.4	481.6	2711.7	4963418.6

3.4 Abstractions

This section comprises the Static Cyber Attacker and Defender abstractions.

3.4.1 Static Cyber Attacker Abstraction

look (e.g., list directory contents or display network configuration)

transfer (e.g., upload or download)

move (e.g., change directories or relocate a file system artifact)

execute (e.g., run a program)

cleanup (e.g., delete a file system artifact or stop a process)

3.4.2 Static Cyber Defender Abstraction

individual research [e.g., studying network (.pcap), host (.evtx) and open (Internet) sources]

annotate [e.g., update Sandia Cyber Omni Tracker (SCOT)]

collaborate (e.g., Skype)

remediate (e.g., implement firewall rule)

3.5 Ontologies

This section comprises the self loop ontology, forward progress ontology and ontology related algorithms.

3.5.1 Self Loop Ontology

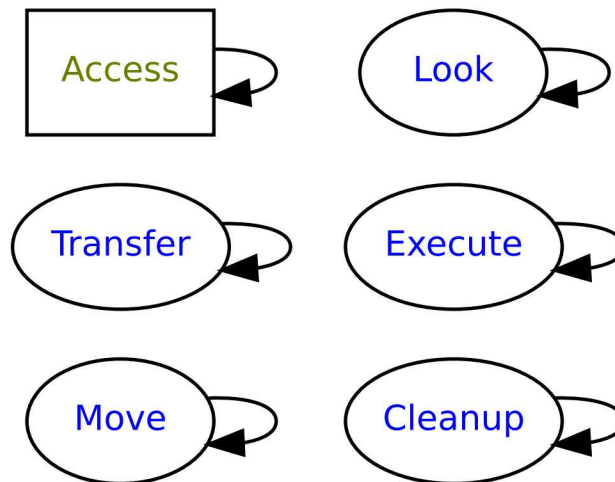


Figure 3.1. Self loop ontology.

The self loop ontology is a simple, straightforward rule set that is useful for framing examples and bootstrapping an ontology refinement activity. For these purposes, the null (no interclass relations) and complete (all interclass relations) are alternatives to the self loop ontology. Figure 3.1 illustrates the self loop ontology. The idea is that activity of one abstraction class is related to other activities of the same class.

3.5.2 Forward Progress Ontology

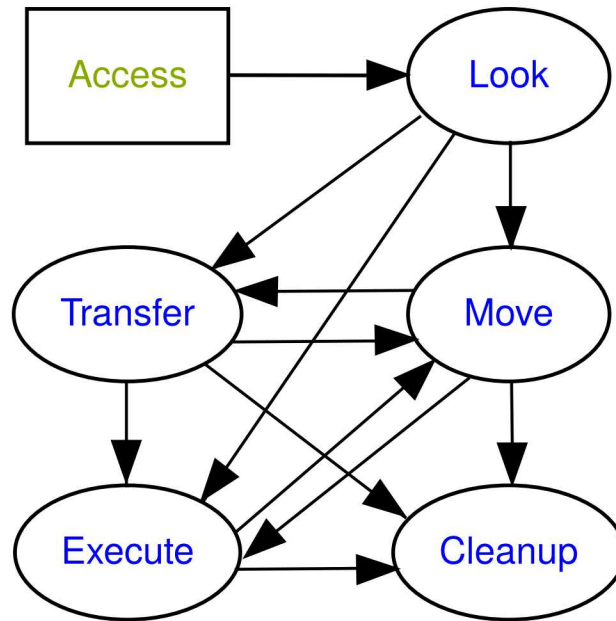


Figure 3.2. Forward progress ontology.

The forward progress ontology was created to track the steps followed by a typical attacker when making progress toward an objective and cleaning up. The ontology has links only for transitions that indicate that the attack is progressing. There are no self-loops, because repeated attack steps could be seen as “compressible” or not the optimal sequence of steps. For example, from a Look the attacker could progress through a Transfer, Execute, or Move step. Likewise, a Look followed directly by Cleanup would not indicate forward progress. Figure 3.1 illustrates the forward progress ontology.

3.5.3 Algorithms

In an ideal world, the human cyber shopper would produce an ontology that perfectly characterizes the actions of any cyber attacker. Furthermore, the cyber threat would be homogeneous (with all attackers behaving the same) and static (where adversary behavior does not change over time).

Unfortunately, the first proposition is unlikely, and the second is naive. While cyber shoppers may struggle to produce the perfect ontology, they can produce accurate linkographs. The algorithms in this subsection provide techniques to transform a linkograph into an ontology (extraction) and, based on linkographs, polish a working ontology (refinement). The practitioner can bootstrap the refinement algorithms we propose with an arbitrary ontology: for example, the self loop, null or complete ontology or a cyber shopper’s best effort at the perfect ontology (e.g., forward progress). From the beginning, our goal was to find a refinement algorithm. However, an extraction algorithm was a clear step towards that goal because the formulation is simpler; specifically, extraction requires fewer parameters.

Ontology Extraction

We describe the formulation for an extraction algorithm as: $L \rightarrow O$.

Simple Extraction Simple extraction is the most naive of approaches. The basic idea is to begin with the null ontology, iterate over every link in the linkograph and add the link’s relation to the ontology if it is not present.

Threshold Extraction Threshold extraction improves on simple extraction: this algorithm only adds the link’s relation to the ontology if it is present in the linkograph more often than not. For example, if there are five opportunities to link an A to a B class node in a linkograph, and three or more of those links are present, then the threshold extraction algorithm adds the relation to the ontology.

Ontology Refinement

We describe the formulation for an extraction algorithm as: $L, O \rightarrow O'$. There are two metrics of interest with this formulation: accuracy and similarity. Accuracy measures how closely a linkograph derived from an ontology matches the precursor linkograph. Similarity measures how closely a precursor ontology (O) matches an output ontology (O'); we calculate this measurement as $1/(1+L(O, O'))$ where $L()$ is modified (inserts and deletes only) Levenshtein distance. We seek to maximize accuracy and similarity. We approach this multi-objective optimization problem by setting a threshold for one metric (similarity) and maximizing the other (accuracy).

Greedy Refinement The greedy refinement algorithm is iterative. During each iteration, the algorithm determines the best rule to add to and the best rule to remove from the ontology. Unless both of these options lower the accuracy of the ontology, the algorithm then adds or removes the better of the two. The greedy refinement algorithm repeats this process until the ontology does not change during an iteration. It is an open question whether or not this algorithm is guaranteed to converge.

Brute Force Minimum Similarity The brute force minimum similarity algorithm is recursive. If the maximum change parameter is greater than zero, the algorithm iterates over all rules. During each iteration, the accuracy of the ontology with the rule added to or removed from the ontology is calculated, and the algorithm is invoked with the provisional ontology and maximum changes decremented. Unless both of these accuracies are lower than the best accuracy, the best ontology and accuracy are replaced by the better of the two. Appendix A contains pseudocode for this algorithm.

High Impact First Minimum Similarity The basic idea is to improve brute force minimum similarity by trying high-impact rules first. We sort the label pairs based on applicability/frequency and iterate over them in decreasing order. Appendix A contains pseudocode for this algorithm.

Windowed Refinement The basic idea is to consider all length k substrings of the input linkograph. For each sublinkograph, we add or subtract at most one rule. This is the one rule that increases the accuracy of the notional derived linkograph the most. If there is no rule that increases the accuracy of the notional derived linkograph, the ontology is not changed based on that sublinkograph.

Converging Windowed Refinement The basic idea is to invoke the windowed refinement algorithm repeatedly, using the results from the previous run, until the results do not change. It is an open question whether or not this algorithm is guaranteed to converge.

3.6 Protocol Analysis

This section comprises our protocol analysis method and results.

Generally speaking, protocol analysis is the practice of collecting a raw data set, organizing it into a dialog transcript and transforming it into an abstract form, such as a linkograph, from which to extract information. Many disciplines have used this technique to characterize individual or group activities.

To perform the protocol analysis of a Windows console session, an analyst began with the first entry and searched the rest of the session for another entry with the same command (e.g., `dir`). Then the analyst searched the rest of the session for another entry with one of the same parameters (e.g., `http://www.foo.com/`). The analyst repeated these two searches for each subsequent entry.

As expected, we found that the time required to apply this technique grew as the square of the number of entries. While it takes seconds for an analyst to extract the linkograph for a session with a few entries, it takes hours for an analyst to extract the linkograph for a session with one hundred entries.

We performed a protocol analysis on all of the Windows console sessions in an example cyber attack dataset. We used these linkographs to exercise the ontology refinement algorithms we discussed previously in this chapter.

3.7 Machine Learning

This section comprises features, behavior prediction and dimension reduction. We used ten fold cross validation in determining all accuracies.

3.7.1 Features

Our machine learning models consider 25 features: node count, critical node count, Cartesian summary (mean, sum and range for x and y), percentage of links (actual link count / possible link count), entropy, T-complexity, link index (link count / node count), graph differences, entropy deviation, mean link coverage, top cover, session start time (of day), session length, mean command delay and behavior ratio (one each for access, look, transfer, move, execute and cleanup).

15 of these features are linkograph metadata, and ten come directly from the raw data.

All but four of these features are continuous. This informs the set of applicable machine learning techniques.

3.7.2 Behavior Prediction

We created machine learning models to predict the next behavior of a cyber attacker. Specifically, we instrumented multinomial classifiers using logistic regression, linear discriminant analysis, K-nearest neighbors (KNN) and support vector machine techniques. For a given behavior, the accuracies of all five techniques were similar. However, for a given technique, the models predicted each of the five behaviors with different accuracies. While the most accurate technique, KNN, predicts cleanup actions with 98.3% accuracy, it predicts look actions with only 85.7% accuracy. Figure 3.3 visualizes these results.

3.7.3 Dimension Reduction

Using the accuracies from the earlier behavior prediction work for supervision, we pursued dimension reduction. There are two basic approaches to dimension reduction: feature selection and feature extraction. We pursued feature selection because it is more intuitive and allows the practitioner to cull the raw data set. We chose two feature selection techniques: forward and backward selection. Forward selection suggests using three predictors for look (look ratio, move ratio and t

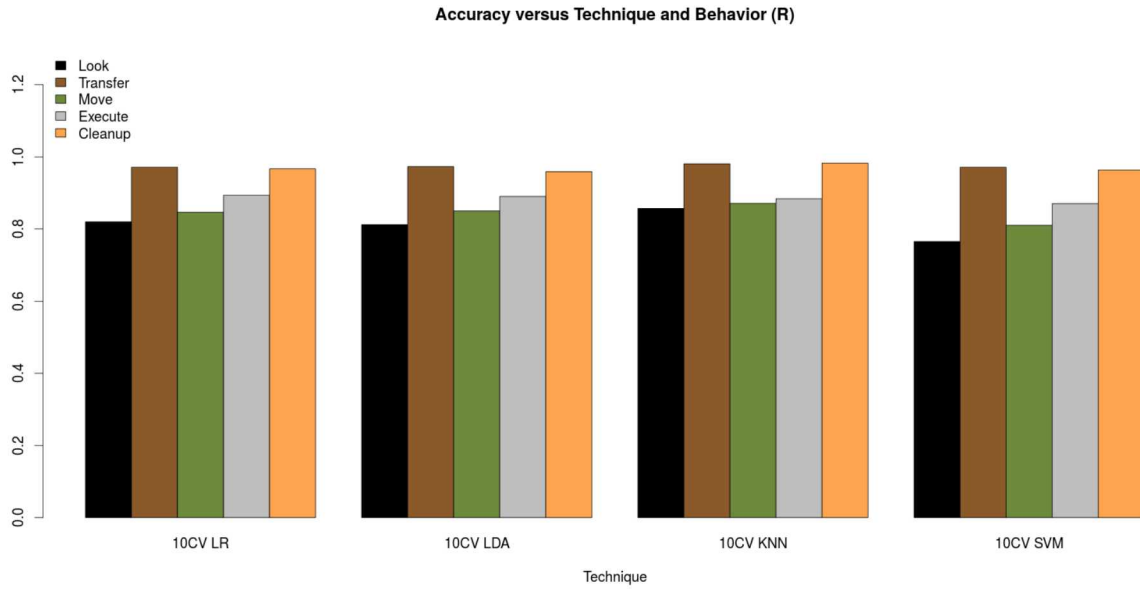


Figure 3.3. Behavior prediction results.

complexity) prediction, one predictor for transfer (node count), move (move ratio), execute (node count) and cleanup (mean link coverage) prediction. Figure 3.4 visualizes these results.

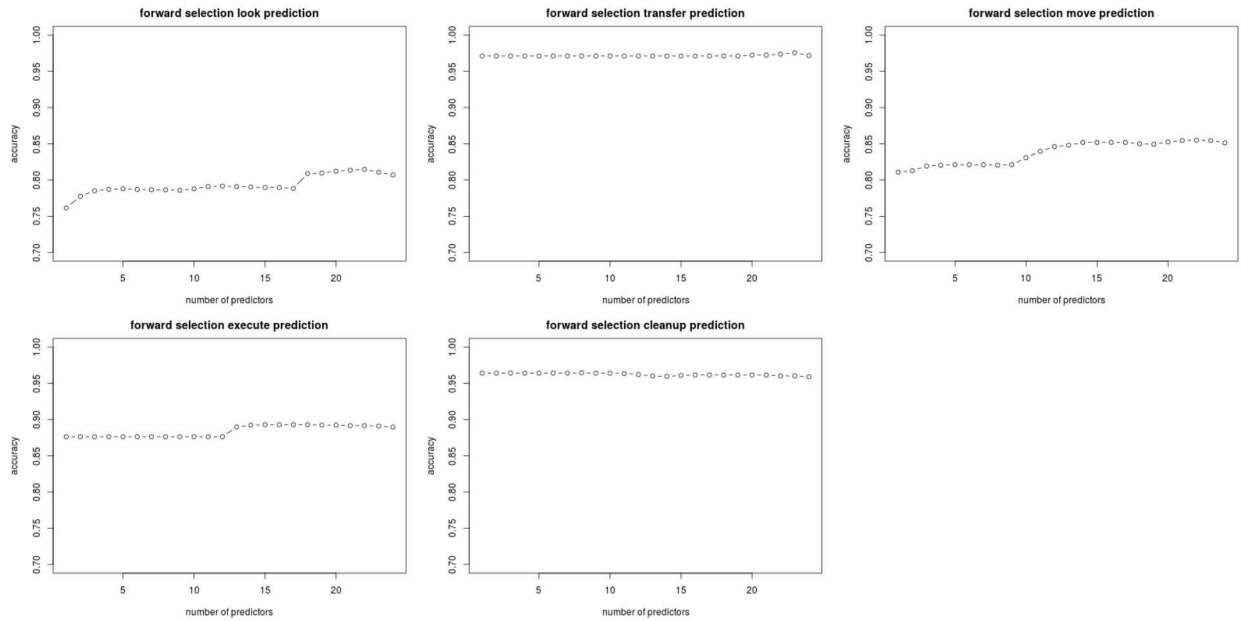


Figure 3.4. Forward selection results.

Backward selection suggests using seven predictors for look (node count, critical node count, $\bar{x}$, $\sum x$, range of x , $\bar{y}$, $\sum y$) prediction, one predictor for transfer (node count), execute (node count) and cleanup (critical node count) prediction and four predictors for move (node count, critical node count, $\bar{x}$ and $\sum x$) prediction. Figure 3.5 visualizes these results.

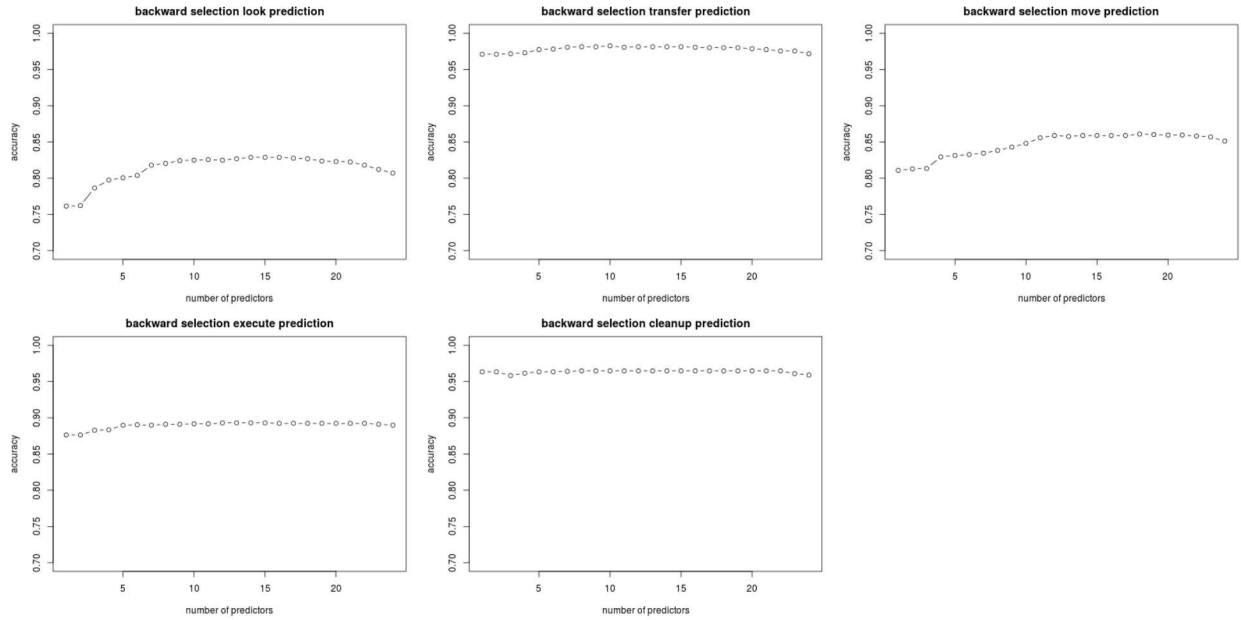


Figure 3.5. Backward selection results.

3.8 Game Theory

3.8.1 Motivation

We pursue a game theoretic approach to cyber shopping for two reasons: First, we want to inform our decision whether or not to apply a countermeasure. This decision decomposes into two sub-problems: identifying critical steps which are opportunities to apply countermeasures and choosing the countermeasure to recommend or schedule. In a parallel effort, the countermeasure framework is being specified and designed. When that work is stable, we will apply our game theoretic model to autonomously schedule or interactively recommend countermeasures. Second, we want to improve our behavior prediction accuracy; in turn, this will increase our ability to anticipate the attacker's next move; if we face a sophisticated adversary, they will likely play the optimal strategy. We will use a hierarchical approach to predict adversary behavior. One hierarchical approach is to trust the prediction of the machine learning model if it is sufficiently confident. Otherwise, we will revert to what the optimal strategy predicts the adversary will do. Another hierarchical approach is to use machine learning to predict the next move and then use the game theoretic model to deter-

mine whether or not to apply a countermeasure. Finding Nash equilibria (optimal strategies) will advance both of these goals by identifying the best plays for the defender and attacker.

3.8.2 Components

The four components of a game are players, information, actions and payoffs. Players are the autonomous actors in the game; they choose actions based on information and realize a payoff at the end. The attacker and the defender are the players in our game. Information provides the players some criteria upon which to choose an action. The attacker knows the address (layer two and three), platform (hardware, OS and applications) and state (registry, file system and database) of defender nodes. The defender knows the attacker actions, the source of attacker input and the destination of exfiltration. Actions are the mechanism by which the players advance the game. The attacker may look, transfer, move, execute or cleanup. The defender may attempt to disrupt the attacker or remain passive. Disrupting the attacker means interfering with the attacker action; this includes denial, delay or corruption. Payoffs characterize the success of each player at the game. Our conceptual model of the game is an attacker moving through a maze. A positive payoff for the attacker corresponds with shortening the distance between the attacker and the exit of the maze. On the other hand, a positive payoff for the defender corresponds with increasing the distance between the attacker and the exit of the maze.

3.8.3 Game Type

We propose a non-cooperative, asymmetric, zero-sum, sequential, imperfect information game. Due to the lack of mutual trust or a trusted agent, the game is non-cooperative. The set difference between the attacker and defender actions provides the asymmetry of the game. The game is zero sum because the shared payoff resource is the progress the attacker makes toward their final goal: One step forward for the attacker is one step closer to failure for the defender. Relaxing this constraint to constant-sum or abandoning it completely is a real possibility. The game is sequential because the defender's only method to act is in countering (or not) any move the attacker makes. Although the attacker may be able to defeat some of the defender's deception, they will not do so all of the time. This is an imperfect information game because at least some of the time, the attacker will not act with complete knowledge of the defender's actions.

3.8.4 Instrumentation

It is not usual for game theoretic models to have a multicomponent payoff for each player. There are two components to payoff: progress units (PU) and intelligence units (IU). PUs indicate progress toward completing the information operation (IO). IUs indicate intelligence gained about the adversary. For the least worst strategy, we equate PU and IU to yield:

For the correct attacker action, if the defender passes, the attacker payoff is 1.

For the correct attacker action, if the defender attempts to disrupt and is successful, the attacker payoff is -1.

For the correct attacker action, if the defender attempts to disrupt and fails, the attacker payoff is 1.

For the incorrect attacker action, if the defender passes, the attacker payoff is -1.

For the incorrect attacker action, if the defender attempts to disrupt and is successful, the attacker payoff is -1.

For the incorrect attacker action, if the defender attempts to disrupt and fails, the attacker payoff is 1.

The following figure illustrates this formulation instrumented in Gambit: Red, blue and green artifacts correspond with the Attacker, Defender and Chance, respectively.

3.8.5 Results

Gambit offers several techniques to compute equilibria. This section summarizes the results of each.

`gambit-enumpure` yields five Nash Equilibria within two seconds. There is one for each of the attacker choices: the defender should disrupt if the attacker made the correct choice and pass otherwise.

`gambit-enumpoly` did not complete after twenty four hours.

`gambit-enummixed` did not complete after twenty four hours.

`gambit-gnm` yields multiple Nash Equilibria after 72 minutes. The format is strange (like `gambit-ipa`).

`gambit-ipa` produces a strange format (like `gambit-gnm`) within two seconds.

`gambit-lcp` yields one Nash Equilibrium in 13 seconds. The attacker should choose cleanup and the defender should disrupt if the attacker made the correct choice and disrupt otherwise.

`gambit-lp` yields one Nash Equilibrium within a second. The attacker should choose look and the defender should choose pass; this is non-intuitive.

`gambit-liap` yields no equilibria in 11 seconds.

`gambit-simpdiv` did not complete after twenty four hours.

`gambit-logit` yields many equilibria within two seconds. The equilibria seem to converge with the attacker choosing each action with probability 0.2 and the defender always disrupting if the

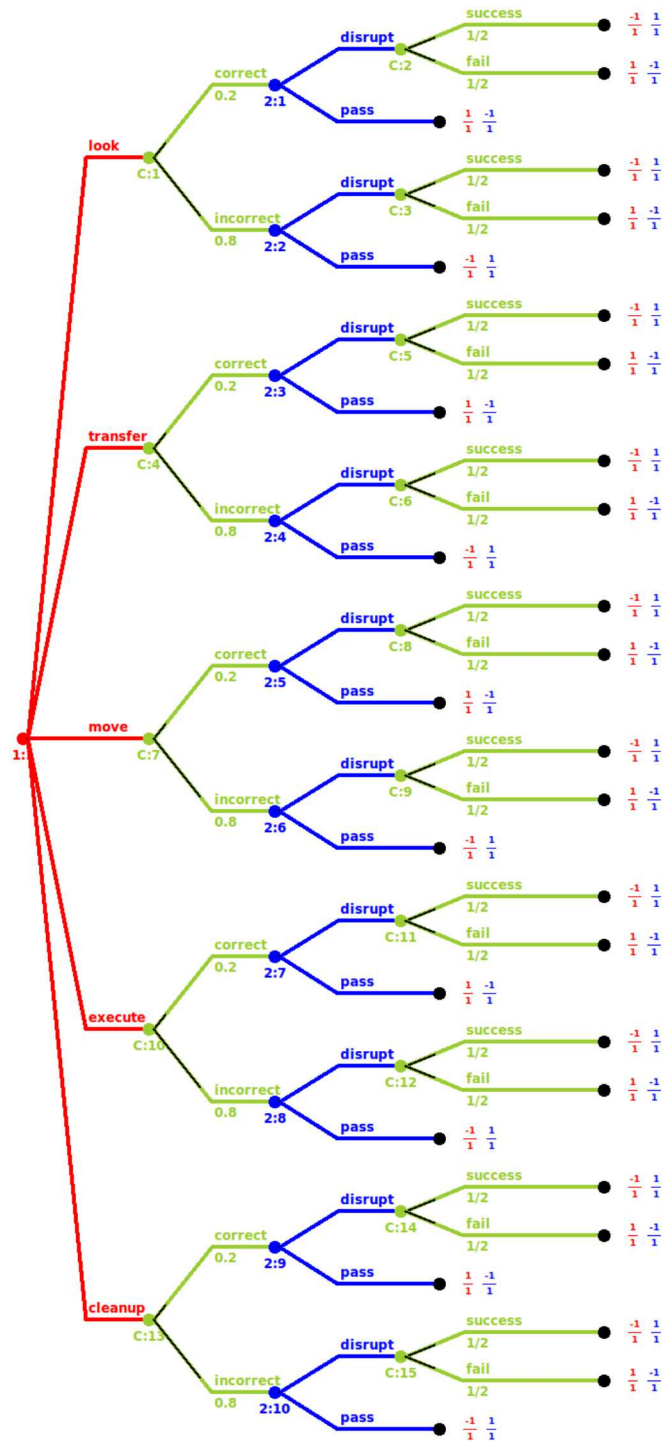


Figure 3.6. Cyber shopping game.

attacker makes the correct choice and passing otherwise.

3.9 Countermeasures

We propose that good countermeasures do two essential things: First, they tease out threat intelligence by prompting the attacker to show their techniques and tools. Second, good countermeasures waste the adversary's resources: for example, operators having to make additional moves, analysts studying bogus products and computers cryptanalyzing random data.

We propose that a countermeasure can go wrong for one of two reasons: First, generating or deploying it wastes defender resources without providing any benefit. Second, a bad countermeasure will spook the adversary and might alert them to being instrumented.

3.9.1 Generating Countermeasures

We propose the following list of countermeasures as a starting point for a future research project or implementation challenge:

file deletion implementation will actually snapshot and delete

network disconnect implementation could involve `net use /delete`

privilege reduction implementation could involve `icacls`

console delay introduce artificial delays into the console session—implementation could involve `wmic process setpriority`

file substitution more complicated, but greater potential confusion—if we anticipate the attacker is about to download or read a file, we snapshot and replace it with something else—this could be context sensitive: detect the file type and use an appropriate replacement—the replacement could be canned or generated dynamically

corrupt directory listing implementation could involve replacing `dir` to a modified version that hides or changes the apparent size of the file or directory of interest

3.9.2 Countermeasure Effectiveness

We propose the following measures of countermeasure effectiveness:

1. A longer than normal delay before the next command could indicate confusion.

2. The repetition of an earlier command may indicate the adversary reorienting themselves.
3. A disoriented adversary may retreat (e.g., backing up a directory or logging out of a host).
4. Selection of a countermeasure that specifically cannot be “burned” simply by revealing its use to the adversary.

There are a number of things that concern the adversary that it is not clear how to measure:

1. Institutional attribution
2. Personal attribution: “I don’t want to be extradited!”
3. Losing a foothold (persistence)
4. Changing terrain (e.g., due to moving target defense or evolution)

This page intentionally left blank.

Chapter 4

Accomplishments

Important measurements of basic research effectiveness are received citations, peer-reviewed publications and delivered seminars. Generated artifacts can indicate the effectiveness of applied research. Formed relationships are germane to both. We hope to receive many citations in the future, but presently, this chapter discusses the remaining four metrics: papers written, presentations given, artifacts generated and collaborations started.

4.1 Papers Written

4.1.1 Communications and Network Security (CNS) 2015

Andrew Fisher and John Jarocki traveled to Florence, Italy in September 2015 to present “Using Linkograph to Understand Cyberattacks.” This paper [5] provided an overview of linkography applied to cyber security, a cyber security abstraction and an ontology based on that abstraction to derive linkographs. We believe this is the first paper in the literature to apply linkography to cyber security.

4.1.2 Computing and Communication Workshop and Conference (CCWC) 2017 (Submitted)

We submitted “Linkography Ontology Refinement and Cybersecurity” to CCWC 2017. If accepted, Robert Mitchell will travel to Las Vegas, Nevada in January 2017 to present it. This paper [19] provided algorithms for refining an ontology based on human-corrected linkographs. The examples in this paper dealt with cyber security in particular, but the results are germane to the linkography community at large.

4.2 Presentations Given

4.2.1 CyberSci 2015

The technical program committee of CyberSci 2015 invited Robert Mitchell to present his recent work. Robert traveled to Fairfax, Virginia in October 2015 to present “Dimension Reduction for Cyber Attack Detection and Analysis.” This talk summarized the classification, dimension reduction and clustering work the team accomplished with regard to predicting the behavior of adversaries and generating abstractions.

4.2.2 MITRE Denial and Deception for Cyber Defense Workshop

Robert Mitchell traveled to McLean, Virginia in July 2016 to present a poster titled “Linkography Based Cyber Security.” This poster covered our dynamic adversary modeling with a human in the loop, technology transfer plan, machine learning results, ontology refinement algorithms and subsessionization techniques, as well as linkography basics.

4.2.3 CyberSci 2016 (Accepted)

Robert Mitchell will travel to Fairfax, Virginia in October 2016 to summarize the results of this LDRD.

4.3 Artifacts Generated

4.3.1 LinkShop

LinkShop is our name for the collection of python libraries, command line utilities, and web application code that we created to perform experiments and produce results for this project. While this is clearly a researcher playground, we also believe the LinkShop tools to be directly useful to cyber analysts for developing models and visualizing attack data.

Our Python linkography package is available in our `linkshop` Git repository. Readers should contact the authors to gain access to this repository.

`grrcon_example`

The LinkShop web application is an interactive tool for dynamically creating, modifying and analyzing linkographs. The tool provides functionality for reading JSON formatted data, labeling commands according to provided abstraction rules, creating linkographs via an initial ontology, and analyzing linkographs using metrics like Shannon entropy. The analyst can then manually add and remove links in linkographs. A new ontology can be extracted automatically or modified directly by the analyst.

Linkshop is built using python3, ThriftPy, Node.js with Express, and D3.

`grrcon_example`

Contains the supporting files for running an example using data from the 2012 GrrCON Network Forensics Challenge `grrcon-encoded.json` to an `svg-html` and `postscript` representation. More information about the example can be found in the `exampleReadme.txt` in this folder.

`hackTestFiles`

These are scripts and files that were used to test `linkoCreate.createLinko`, `linkoCreate.writeLinkoJson`, `linkoCreate.readLinkoJson` and `labels.labelCommands`. They are not official tests.

`labels.py`

`labelCommands` takes a JavaScript Object Notation (JSON) command file and a JSON label file and creates a label object and optionally a JSON file. See the docstring for more details.

`linkoCreate.py`

`Linkograph` is the class definition of the linkograph object. A `Linkograph` extends a list to allow for adding attributes. The only default attribute is a list of the labels.

`writeLinkoJson` handles writing a `Linkograph` to JSON files.

`readLinkoJson` handles reading a `Linkograph` from a JSON file.

`readLinkoCSV` creates a `Linkograph` from a CSV file.

`createLinko` creates a `Linkograph` from a label object JSON and a rule JSON.

linkoDraw.py

linkoDrawEPS creates a postscript file from a Linkograph.

linkoSVG.py

linkoDrawSVG creates a Scalable Vector Graphics (SVG) representation from a Linkograph.

markov.py

createMarkov creates a Markov model of the linkograph.

runTest.py

This is a script for running the unit tests.

stats.py

Statistics functions for analyzing linkographs.

totalLinks gives the number of possible links for a linkograph.

totalLabels gives the frequency of each label.

percentageOfEntries gives the percentage of lines with each label.

links counts the number of links.

linkCount counts the number of links in a list passed. This function is a helper function for other functions.

linkTotal calculates the total possible links along a line. This function is a helper function for other functions.

percentageOfLinks gives the percentage of links.

graphEntropy gives the Shannon Entropy for the linkograph.

shannonEntropy is the bare Shannon entropy function.

linkEntropy calculates forelink and backlink entropies.

4.3.2 Machine Learning Models

Our machine learning models are available in the `machineLearning` directory of our Git repository. Readers should contact the authors to gain access to this repository.

4.4 Collaborations Started

1. Massachusetts Institute of Technology (MIT) Lincoln Laboratory's Cyber Systems and Operations Group
2. Army Research Lab (ARL)
3. Department 1463 Cognitive Sciences and Systems
4. Department 6114 Military and Energy Systems Analysis
5. Department of the Navy
6. Naval Research Laboratory (NRL)
7. MITRE

Robert met Alexia Schulz and Michael Kotson from Massachusetts Institute of Technology (MIT) Lincoln Laboratory's Cyber Systems and Operations Group at CyberSci 2015. Afterwards, Alexia invited Robert to give a talk at MIT Lincoln Laboratory. John and Andrew met Michael Kotson previously at CNS 2015. MIT Lincoln Laboratory's Cyber Systems and Operations Group is a potential research partner.

Robert met Kyra Comroe and Lawrence Knachel from Army Research Lab (ARL) at CyberSci 2015. John met Kyra Comroe at a technical exchange meeting shortly before CyberSci 2015. ARL is a potential research partner and technology transfer recipient/benefactor.

Asmeret Bier from Department 1463 was internally referred to our LDRD to enhance her work. This collaboration resulted in an FY 2017 LDRD idea (with Nadine Miner). Asmeret is a potential research partner.

Nadine Miner from Department 6114 was internally referred to our LDRD to enhance her work. This collaboration resulted in an FY 2017 LDRD idea (with Asmeret Bier). Nadine is a potential research partner.

Robert met Daniel Ricci from the Department of the Navy at the MITRE Workshop. The Department of the Navy is a potential technology transfer recipient/benefactor.

Robert met Joseph Langley from Naval Research Laboratory at the MITRE Workshop. Joseph sits at the Joint Base Anacostia-Bolling location and is a potential research partner.

We had multiple discussions with Adam Pennington from MITRE over the course of three years that helped inform our perspective, and we passed him a copy of the CNS paper in 2015.

Chapter 5

Conclusion

5.1 Lessons Learned

Our team learned many lessons during our three year journey. This caused us to realize that our task list continued to “expand at the middle” over the lifetime of the project. Eventually we decided to create the “system” diagram as illustrated earlier in this report to capture what we worked on, continue to work on, and know we need to do but have yet to start.

A lesson that appears obvious now, in hindsight, is that applying a method, such as linkography, from a completely different research domain requires taking significant time to align terminology and methods to a new domain. In our case, we needed to provide some terminology and formal definitions to enable researchers to utilize linkography for cyber attack analysis. Since we want to do this work at scale, in an automated fashion, we also needed to create software to ingest data, normalize it to JSON, produce results, and provide a user interface for researchers to interact with. Because we have tried very hard to keep all of our terminology, methodology, and code as general as possible, we believe the result is a useful to the wider linkography community as well as cyber security.

Existing models of cyber attack behavior focus on high level activities such as reconnaissance, exploitation, delivery, or command and control. The classes of actions at this level of abstraction encompass entire ecosystems of activity. As such, they are not close enough to the real data for us to make important, real-time decisions. In fact, we learned quickly that human defined abstractions often do not even maintain a constant level of abstraction. Since we did not want to restrain future research, even our own, to a static level or set of abstractions, we decided to create a system that is flexible enough to operate at whatever level is appropriate.

With our gained insights into the power of adding flexibility, we came to the conclusion that a way to “evolve” the artifacts of linkography was also needed: namely the abstraction and the ontology. This thread of research led us to discover multiple ways to refine an existing ontology until it matched a provided data set. True to our objectives of creating a transparent analytic system that allowed a human-in-the-loop, our method has the ability to learn from the input of a human analyst. We also identified the need to create a similar process for evolution of the abstraction classes, but our time ran out to tackle this body of work.

5.2 Future Work

We ended our subsessionization investigation with substringing techniques which allows us to break the input from one actor into objectives. In order to deconflict simultaneous actors, we will pursue subsequencing approaches to subsessionization.

Our current body of work assumes the input flow comes from a malicious user. We will provide linkograph based intrusion detection in order to filter data from legitimate users.

We will aggregate alerts using machine learning classification or clustering models based on linkograph features. This will benefit the workflow of incident responders by organizing their data.

We will create honeycues to enable cyber deception operations.

Using Bomb Damage Assessment as a metaphor, we will measure the effectiveness of honeycues. This data will feed game theoretic modeling we will continue.

Our current body of work considers Windows console data. We believe our results are generalizable and will apply linkography to other data sets (e.g. web server and database logs).

We will perform protocol analyses in support of cyber security training (e.g., TracerFIRE RE-COIL) and create a cyber security decision support agent.

References

- [1] Niccol Becattini, Gaetano Cascini, and Federico Rotini. An OTSM-TRIZ Based Framework Towards the Computer-Aided Identification of Cognitive Processes in Design Protocols. In John S. Gero and Sean Hanna, editors, *Design Computing and Cognition*, pages 99–117. Springer International Publishing, 2015.
- [2] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [3] Tamer El-Khouly and Alan Penn. On an integrated analytical approach to describe quality design process in light of deterministic information theory. In *Proc. of DCC*, 2012.
- [4] Tamer El-Khouly and Alan Penn. Order, Structure and Disorder in Space Syntax and Linkography: Intelligibility, Entropy, and Complexity Measures. In *Proc. of SSS*, 2012.
- [5] Andrew Fisher, Kent Carson, David Zage, and John Jarocki. Using Linkography to Understand Cyberattacks. In *IEEE Conference on Communications and Network Security*, Florence, Italy, September 2015.
- [6] Andrew Fisher, David Zage, and John Jarocki. Towards a Formalization of Linkography. Technical Report SAND2016-8942R, Sandia National Laboratories (SNL), Albuquerque, NM, USA, 2016.
- [7] Renate Fruchter. Bricks & bits & interaction. In *New Frontiers in Artificial Intelligence*, pages 35–42. Springer, 2001.
- [8] John S Gero. Design prototypes: a knowledge representation schema for design. *AI Magazine*, 11(4):26–36, 1990.
- [9] John S Gero, Jeff WT Kan, and Morteza Pourmohamadi. Analysing design protocols: Development of methods and tools. In *Proceedings of the 3rd International Conference on Research into Design Engineering*, Bangalore, India, January 2011.
- [10] Gabriela Goldschmidt. The designer as a team of one. *Design Studies*, 16:189–209, 1995.
- [11] Gabriela Goldschmidt. *Linkography: Unfolding the Design Process*. The MIT Press, 2014.
- [12] Jeff W.T. Kan, Zafer Bilda, and John S. Gero. Comparing entropy measures of idea links in design protocols. In *Design Computing and Cognition*. Springer, 2006.
- [13] Jeff W.T. Kan and John S. Gero. Can entropy represent design richness in team designing? In *Proc. of CCAADRIA*, 2005.

- [14] Jeff W.T. Kan and John S. Gero. Design behaviour measurement by quantifying linkography in protocol studies of designing. *Human behaviour in designing*, 5:47–58, 2005.
- [15] Jeff W.T. Kan and John S. Gero. Acquiring information from linkography in protocol studies of designing. *Design Studies*, 29:315–337, 2008.
- [16] Jeff W.T. Kan and John S. Gero. *About: Designing-Analysing Design Meetings*, chapter Using the FBS ontology to capture semantic design information in design protocol studies, pages 213–229. CRC Press, 2009.
- [17] Nikolai Khomenko and Dmitry Kucharavy. OTSM-TRIZ problem solving process: solutions and their classification. In *Proceedings of TRIZ Future Conference*, pages 6–8, Strasbourg, France, November 2002.
- [18] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.
- [19] Robert Mitchell, Andrew Fisher, Scott Watson, and John Jarocki. Linkography Ontology Refinement and Cybersecurity. In *Computing and Communication Workshop and Conference*, Las Vegas, NV, USA, January 2017.
- [20] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.
- [21] Alan Penn. Who enjoys shopping in IKEA? <https://youtu.be/NkePRXxH9D4>. Accessed: 2016-09-08.
- [22] Morteza Pourmohamadi and John S. Gero. LINKOgrapher: An analysis tool to study design protocols based on FBS coding scheme. In *Proc. of ICED*, 2011.

Appendix A

Appendix: Ontology Refinement Algorithms

A.1 Brute Force Minimum Similarity Pseudocode

Algorithm 1 Brute Force Minimum Similarity

```
1:  $a' \leftarrow \text{calculateaccuracy}(o, lg)$ 
2:  $o' \leftarrow \text{copy.deepcopy}(o)$ 
3: if  $0 < m$  then
4:    $\text{abstractionclasses} \leftarrow \text{createabstractionclasses}(lg)$ 
5:   for  $\text{initial}$  in  $\text{abstractionclasses}$  do
6:     for  $\text{terminal}$  in  $\text{abstractionclasses}$  do
7:        $\text{pair} \leftarrow (\text{initial}, \text{terminal})$ 
8:        $o_0 \leftarrow \text{copy.deepcopy}(o)$ 
9:       if  $\text{initial}$  in  $o$  then
10:        if  $\text{terminal}$  in  $o[\text{initial}]$  then
11:           $o_0 \leftarrow \text{subtractrelation}(o_0, \text{pair})$ 
12:        else
13:           $o_0 \leftarrow \text{addrelation}(o_0, \text{pair})$ 
14:        end if
15:      else
16:         $o_0 \leftarrow \text{addrelation}(o_0, \text{pair})$ 
17:      end if
18:       $a_0 \leftarrow \text{calculateaccuracy}(o_0, lg)$ 
19:       $o_1, a_1 \leftarrow \text{recurse}(lg, o_0, m - 1)$ 
20:      if  $a_0 > a'$  then
21:         $o' \leftarrow o_0$ 
22:         $a' \leftarrow a_0$ 
23:      end if
24:      if  $a_1 > a'$  then
25:         $o' \leftarrow o_1$ 
26:         $a' \leftarrow a_1$ 
27:      end if
28:    end for
29:  end for
30: end if
31: return  $o', a'$ 
```

A.2 High Impact First Minimum Similarity Pseudocode

Algorithm 2 High Impact First Minimum Similarity

```
1:  $a' \leftarrow \text{calculateaccuracy}(o, lg)$ 
2:  $o' \leftarrow \text{copy.deepcopy}(o)$ 
3: if  $0 < m$  then
4:   abstractionclasses  $\leftarrow \text{createabstractionclasses}(lg)$ 
5:   pairlist  $\leftarrow []$ 
6:   for initial in abstractionclasses do
7:     for terminal in abstractionclasses do
8:       pairlist.append((initial, terminal))
9:     end for
10:  end for
11:  sortedpairlist  $\leftarrow \text{sortpairlist}(\text{pairlist}, lg)$ 
12:  for pair in sortedpairlist do
13:     $\hat{o} \leftarrow \text{copy.deepcopy}(o')$ 
14:    if pair[0] in  $\hat{o}$  then
15:      if pair[1] in  $\hat{o}[\text{pair}[0]]$  then
16:         $\hat{o} \leftarrow \text{subtractrelation}(\hat{o}, \text{pair})$ 
17:      else:
18:         $\hat{o} \leftarrow \text{addrelation}(\hat{o}, \text{pair})$ 
19:      end if
20:    else:
21:       $\hat{o} \leftarrow \text{addrelation}(\hat{o}, \text{pair})$ 
22:    end if
23:     $\hat{a} \leftarrow \text{calculateaccuracy}(\hat{o}, lg)$ 
24:    if  $\hat{a} > a'$  then
25:       $m -= 1$ 
26:       $a' \leftarrow \hat{a}$ 
27:       $o' \leftarrow \hat{o}$ 
28:      if  $0 == m$  then
29:        break
30:      end if
31:    end if
32:  end for
33: end if
34: return  $o', a'$ 
```

This page intentionally left blank.

Appendix B

Appendix: Functional Linkography

The current approach to abstraction classes is to label commands according to a set of regular expressions that define membership in a particular category. The labeling process is to determine if one of the regular expressions matches an event and apply the corresponding abstraction class label. Aside from it being tedious to construct the regular expression, this approach has a further drawback of being centered around the notion of string matching. However, other methods of labeling are possible. For example, one could imagine selecting a few key events and giving the same label to every event within a given Levenshtein distance. Another approach is to give the same label to events that process a file in the same directory.

The current approach to ontologies is to consider them graphs with vertices that are abstraction classes and directed edges that are the relations between the abstraction classes. The relations in the ontology determine the linking used to produce a linkograph. For example, a relation $A \rightarrow B$ in the ontology means that every event labeled A is linked to every subsequent event labeled B. This strategy does not allow a fine grained method of determining the links. For example, maybe we only want to link two events that are processing a particular file. It is not easy to accomplish this type of link in the current approach. One technique would be to create an abstraction class that labels every event that processes that file and have a self-loop for that abstraction class.

The basic idea for the new approach is to use functions for determining the abstraction class labeling and functions for determining the linking. Thus, instead of a regular expression that determines which abstraction class an event belongs to, the event is passed to a function that processes it to determine what the appropriate label should be. This function could be anything. For example, the Levenshtein distance method mentioned above can be realized by having a list of the key events and writing a function that determines the distance from them. Every event within a given distance is given the same label.

For ontologies, the function approach would be to allow functions to process the labels for each pair of nodes in the linkograph and then determine whether they should be linked. These functions can be allowed to process any additional type of data that is desired, for example, the file being processed. With this idea, the processed file problem raised above can be solved by using the file being processed as a parameter and linking the nodes with events that process the same file.

B.1 Motivations

The main point of the abstraction class and ontology systems is to determine how to create links in a linkograph. Since the ontology portion is the piece that actually does the linking, this piece should be considered first. In other words, determining the labeling and assigning relations enable the system to determine the linking. So at a higher level, the labeling and relations provide the information needed by the system to allow it to use its linking decision process. From this perspective, we see that the abstraction classes filter the events into a form that the linking mechanism can process to create links. Previously, the events were separated according to different categories of function. However, as illustrated above, it may also be useful to separate the events according to the files that they process. And what should be done with compound events? In order for a system to be able to answer these questions, it needs to be able to break up each event into the appropriate chunks. To separate the events according to different categories of function, we need to split up the event based on the function parameters. For example, the arguments should be separated. To handle labeling according to the files being processed, the event needs to be filtered for any arguments that involve a file. To handle compound events, the input needs to be split up into its constituent events.

We have just described the need to be able to take an event and split it according to things like commands and arguments, files processed and constituent events. Thus, the question is what type of information needs to be separated out from the event in order to enable a subsequent linking function to do its job. So, what is needed first is a parser combined with a collection of data types to tag portions of the event with. For example, to handle the cases up to this point, we would want the parser to split compound events into constituent events, identify the command portion, identify the arguments portion, and determine if the event processes any files. Thus, instead of having a label at the end, we would have an object with tags (possibly a tree of tags) that identify key bits of information like the command, files, network address and so on. A linking function would take this object, filter it according to the type of data that it cares about, and determine whether the link should exist. In the linking on file processed example, the corresponding linking function would filter on files processed and determine if the two events should be linked.

Does this direction mean that abstraction classes are gone as we know them? Not really. Currently the tool can label an event according to a type of command and apply a label based on the file. For example, the process of linking all events that process the same files can nearly be done now. For each file in the sequence of events you are interested in, you write a regular expression that recognizes that file regardless of any other commands or arguments. You then create an ontology that uses self-loops to create relations from these file abstraction classes to themselves. So this proposal is not something new but rather brings to the forefront an idea that was always under consideration. In bringing this idea out, it can become a proper part of the tool flow.

With this more broad notion of abstraction, what we would have is a collection of views of the data that can identify groups of events together according to the content of the tags. So, one view can be to filter on having the same tagged file that the event processes. Another view could filter on just having a file process tag. Hierarchy in the abstractions can be coarser or finer views that separate certain abstraction classes or combine them. In this framework, combining abstraction

classes amounts to combining the functions on the rules and splitting abstraction classes amounts to creating functions that separate the events.

If we use functions to combine and separate abstraction classes, how can we have a system that allows a user to do this? One possible answer comes back to the tag parsing. You could allow a user to identify which tags of an event string they are interested in and allow certain methods for combining them. For example, a user could start with determining that they only care about tags for the command name. Then they combine the tags together by identifying abstraction classes of events they are interested in. Next they see that events of one abstraction class have file redirection and they want to bin them in a different abstraction class. If file redirection is part of the tagging system, the user would identify that they care about the file redirection. In turn they would create a new way of combining the tags by having the rule for one abstraction class include checking for a redirection tag and not consider the event to be in that abstraction class if a redirection tag is found. In addition, the original rule would have the system check for a redirection tag and consider the event some other abstraction class if the redirection command is found. Next, the user may decide that they want to separate the abstraction class into directory structure and network subclasses. They would then change the rule on the events and say that directory-oriented events belong to one abstraction class and network-oriented events belong to another. Perhaps they want to split the network abstraction class according to the which events are looking at the same IP address. The user would then add an additional condition to the rule grouping network events that looks at the network tag and ensures that the network tag is the same. The key for enabling this type of process is to have an underlying system that tags the event appropriately and provides rich enough functions for the user to combine the tags. One of the most important abilities such a scheme requires is to eventually separate every event into its own abstraction class. This ability provides the finest splitting of the events possible, which demonstrates that the linking rules have maximum precision.

By tagging the events and allowing methods to split the events according to these tags, we provide the ability to combine and split abstraction classes that the linking rules process. However, the linking rules can be allowed to process these same tags directly. So, what becomes the point of abstraction classes and how do they relate to the linking rules? In one sense, they can be orthogonal. Abstraction classes can be a way of labeling the events to get a high level understanding on what is going on or to combine and split nodes to abstract or refine the linkograph nodes. On the other hand, abstraction classes can be used to aggregate events to simplify the linking rules. So, if one is writing a linking rule that says, "I want all events of this type to be linked to all events of this other type," then this situation would be a candidate for abstracting these two types of events. So, in relation to linking rules, abstraction classes would be a means of removing the tag information that is unimportant to the linking rule, or to collapse or expand groups of nodes that a user wishes to be considered a single event or several. Thus, in the end, the linking rules also require the events to be tagged in an appropriate way and to have an ability to combine the tags. The results of such rules would be to create a link instead of considering the events to be the same abstraction class.

B.2 Infrastructure

The above motivation introduces the idea of having three main pieces to the construction and presentation of a linkograph: parsing, functional abstraction and functional ontology.

B.2.1 Parsing

The core aspect of the parsing phase is to identify artifacts of events that will be used by the practitioner to define rules for viewing and linking. Its job is to break the events up into immediately accessible chunks that the views and ontology functions can process. For example, a view needs to be able to query the result of the parser for things like the command being used. So this component needs to be an event parser that will create a parse tree of the event that identifies the relevant chunks. It would also be helpful if the parser was adaptable to allow a user to identify new artifacts for recognition. This may be ambitious, but it should be possible for a developer to add additional nodes to the parse tree. A possible direction is to use something like ANTLR [20], which is a system designed to build parsers for a user based on a grammar. It is possible to make this process dynamic, so the user would add a grammar rule, ANTLR would compile a new parser, and the new parser would provide the requested artifacts.

An additional constraint on the result of the parser is that compound events packed as a single event should be separated. For example, if a for loop is identified, the constituent events should be separated out.

B.2.2 Functional Abstraction

The next part is to manipulate how events are grouped, which is identifying abstractions. Abstractions should have three main abilities, to identify all events that meet a certain criteria, to combine events, and to split events that were combined. The first operation is what we do now. The second two operations would change the number of nodes in a linkograph. For example, if you see a particular set of three events together acting on the same file, then you want to identify these events as a single operation. This can be done by identifying that type of operation as one abstraction class, but it is more elegant to combine the events into a single node. In order to accomplish these tasks, the user should have the ability to create an abstraction layer and identify a set of rules based on the tags to identify which events should be combined, split or labeled the same. The system should be able to achieve both extremes of combining all events as a single node and splitting any event into many nodes. Similarly, labeling should be able to go from labeling all events as the same to providing a unique label for each distinct event. At the end, the abstraction layers are essentially decision processes where the user prescribes a set of conditions for event identification.

B.2.3 Functional Ontology

The linking rules should be able to provide links between two events based on the tags and the abstraction layer. They are programmable decision processes. For example, consider two events in the context of a networking abstraction layer: If the first event belongs to one abstraction class and the second event belongs to another abstraction class, then link the two together if they both process file names that are within an edit distance of 3.

This page intentionally left blank.

Appendix C

Appendix: Deriving Markov Models

C.1 Introduction

Using Markov models to help understand the behavior in a linkograph is nothing new. The tool LINKOgrapher [22] can generate both the first order and second order Markov models based on the FBS ontology. Gero, et al. [9] use Markov models to predict the probability of design issues coming after each other in strict time-order sequence. We propose the use of Markov models to model the behavior of the actor producing the events being analyzed.

C.2 Representation

Markov models, specifically first order models, have two common representations: as a graph and as a matrix (called a transition matrix). An example of a graphical representation is the following.

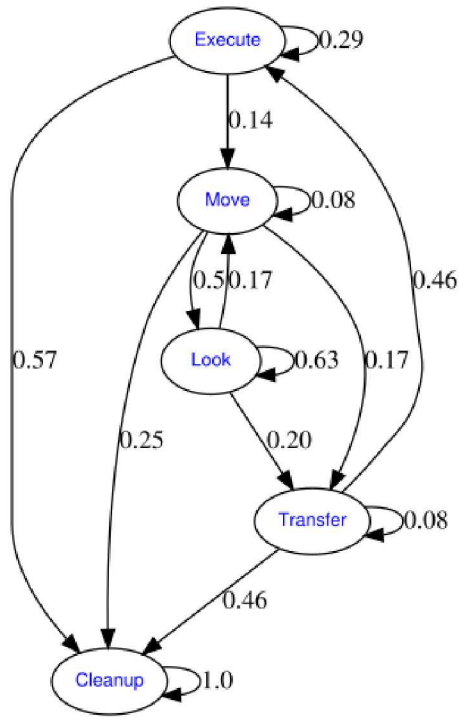


Figure C.1. Markov Model example.

In this sort of representation, the edges give the probability of moving from the present state to the next state indicated by the arrow. So, there is a 57% for moving from state Execute to state Cleanup.

	Cleanup	Execute	Look	Move	Transfer
Cleanup	1.0	0.0	0.0	0.0	0.0
Execute	0.57	0.29	0.0	0.14	0.0
Look	0.0	0.0	0.63	0.17	0.20
Move	0.25	0.0	0.5	0.08	0.17
Transfer	0.46	0.46	0.0	0.0	0.08

In the transition matrix, the row indicate the initial state and the column indicates the terminal state. For example, in row Execute and column Cleanup, the entry is 0.57, corresponding to Execute → Cleanup in the graphical representation.

Often the graphical view is preferred for illustrative purposes and the matrix view is preferred for calculations.

C.3 Markov Interpretation

The LinkShop markov package has two different types of Markov models that it can construct: link-predictor and behavioral. With the link-predictor Markov model, a probability of 0.57 for `Execute`→`Cleanup` means that if the current node has an abstraction class label of `Execute`, then there is a 57% chance that there is a later node labeled `Cleanup`. That is, there is a 57% chance that there is a link connecting the current node with label `Execute` to a later node with label `Cleanup` somewhere in the linkograph. With the behavioral Markov model, a probability of 0.57 for `Execute`→`Cleanup` means that if the current node has an abstraction class label of `Execute`, then there is a 57% chance that the very next node has a label of `Cleanup`.

C.4 Uses

Two of the most obvious uses of Markov models are: as a profile of the transition behavior of a sequence of events and as a method for producing events. The utility of these uses have not been fully explored, yet. However, a couple of observations can be made.

- Using Markov models to profile the next transition is the same as using them to predict the next state. Thus, using Markov models in this way is a less sophisticated approach to finding the next behavior than the Machine Learning approach.
- It is not useful to use Markov models to both produce a sequence of events as actors and use them to profile such sequences of events. This type of use is circular, the profiling Markov model will converge to the producing Markov model.

C.5 Markov Models of Sessions (Link-Predictor Model)

The models in this section were constructed using the LinkShop tool `markov.py`, `dot2svg.py` and Inkscape. For example, the following code takes a linkograph in a file `linko.json`, constructs the Markov model using the link-predictor method, and prints the Markov model as a Graph Description Language (DOT), Scalable Vector Graphics (SVG) and Portable Network Graphics (PNG) file.

```
# Construct a markov model from linko.json and print in dot format
markovMatrix.py -d -m link-predictor -o markovModel.dot linko.json

# Convert the dot version to an svg version
dot2svg.py markovModel.dot markovModel.svg

# Convert the svg to a png
inkscape -e markovModel.png markovModel.svg
```

Figure C.2. Creating a link-predictor model in LinkShop.

Multiple labels are allowed in this example (a condition we later realized should not be allowed). Each of the states that are present in the model represent an abstraction class that appears as a label in the linkograph. Transition percentages are limited to two decimal places.

For the sake of brevity, only the derived models for sessions 1, 32 and 53 are displayed here. The rest are available from the team wiki or by request from the authors.

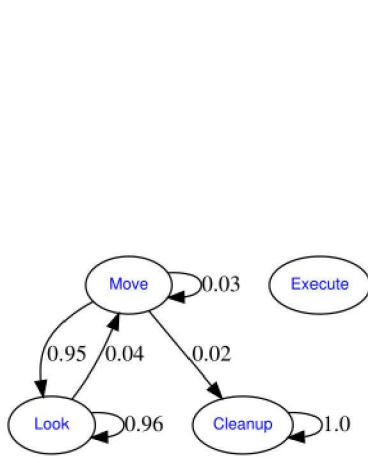


Figure C.3. Session 1.

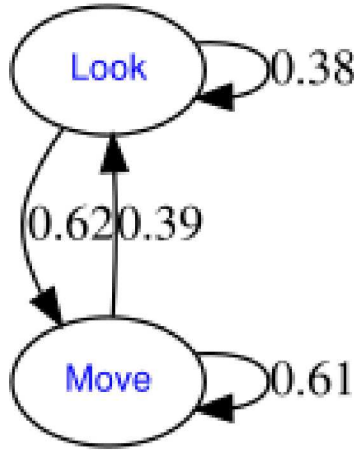


Figure C.4. Session 32.

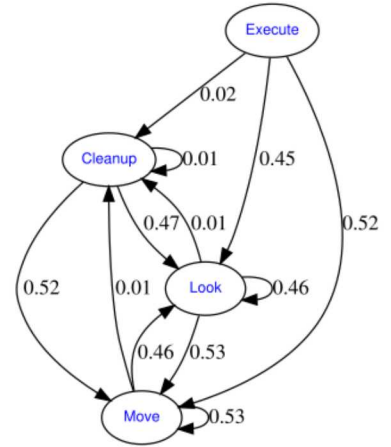


Figure C.5. Session 53.

The overall Markov model is derived from a combination of all the sessions.

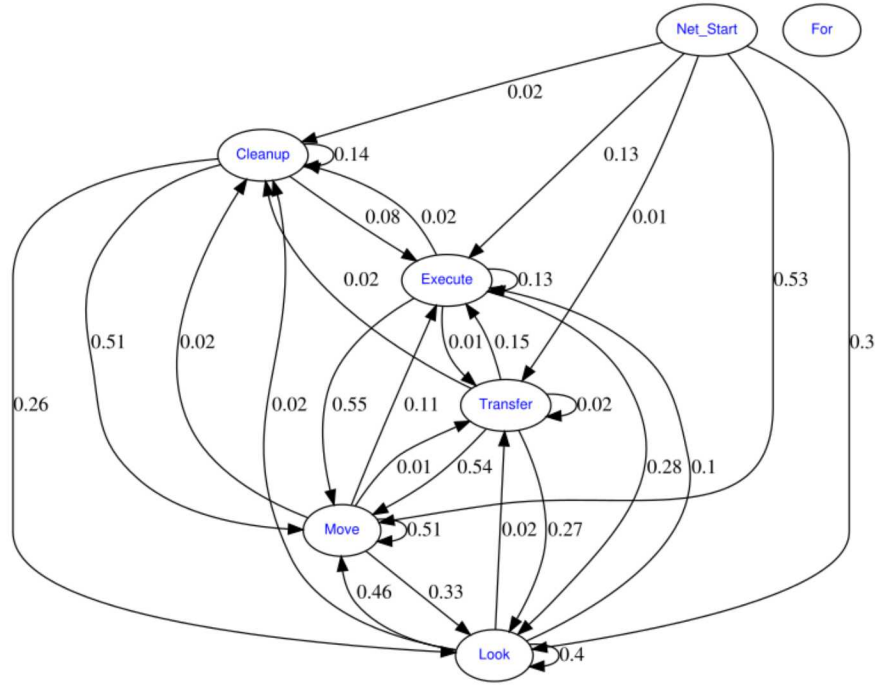


Figure C.6. Overall link-predictor Markov model.

The model can be represented as the following link-predictor transition matrix. The columns and rows are the abstraction classes: Cleanup, Execute, For, Look, Move, Net.Start and Transfer.

$$\begin{bmatrix}
 0.14 & 0.08 & 0.0 & 0.26 & 0.51 & 0.0 & 0.0 \\
 0.02 & 0.13 & 0.0 & 0.28 & 0.55 & 0.0 & 0.01 \\
 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
 0.02 & 0.1 & 0.0 & 0.4 & 0.46 & 0.0 & 0.02 \\
 0.02 & 0.11 & 0.0 & 0.33 & 0.51 & 0.0 & 0.01 \\
 0.02 & 0.13 & 0.0 & 0.3 & 0.53 & 0.0 & 0.01 \\
 0.02 & 0.15 & 0.0 & 0.27 & 0.54 & 0.0 & 0.02
 \end{bmatrix} \quad (C.1)$$

Figure C.7. Link-predictor transition matrix.

C.6 Markov Models of Sessions (Behavioral Model)

The following figures are first order Markov models for the same three sessions examined in the link-predictor section, but constructed using the behavioral model. These models were again constructed using `markovMatrix.py`, `dot2svg.py` and Inkscape. However, this time `markovMatrix.py` was invoked with the argument `-m behavioral`.

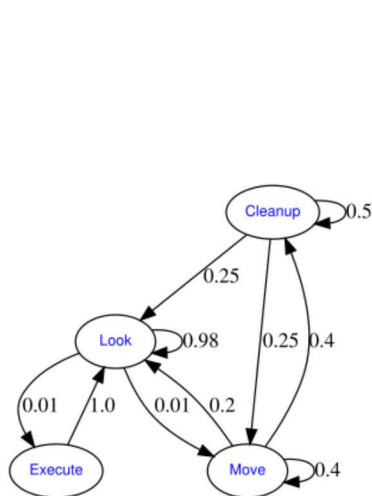


Figure C.8. Session 1.

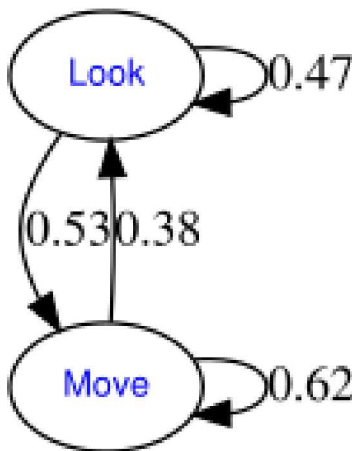


Figure C.9. Session 32.

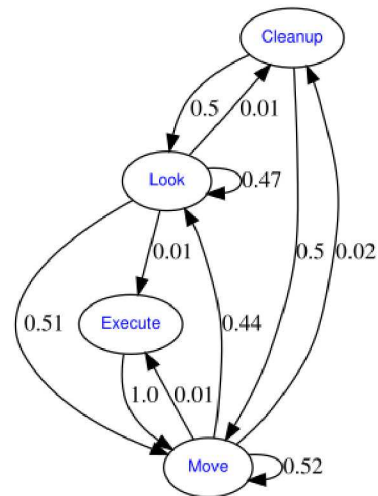


Figure C.10. Session 53.

The following is the behavioral Markov model derived from the combination of all 53 sessions.

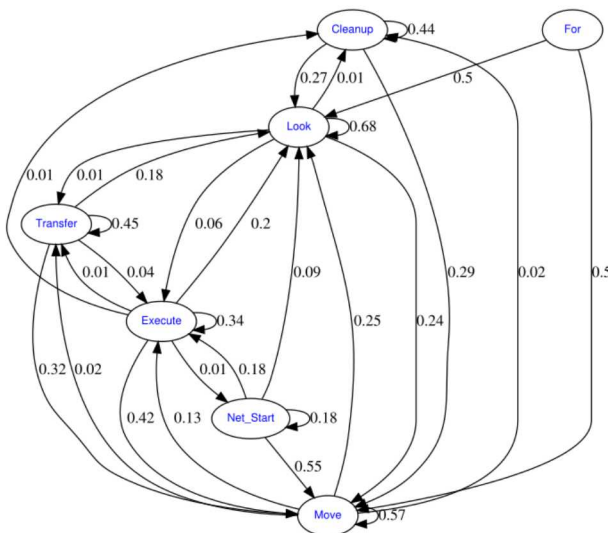


Figure C.11. Overall behavioral Markov model.

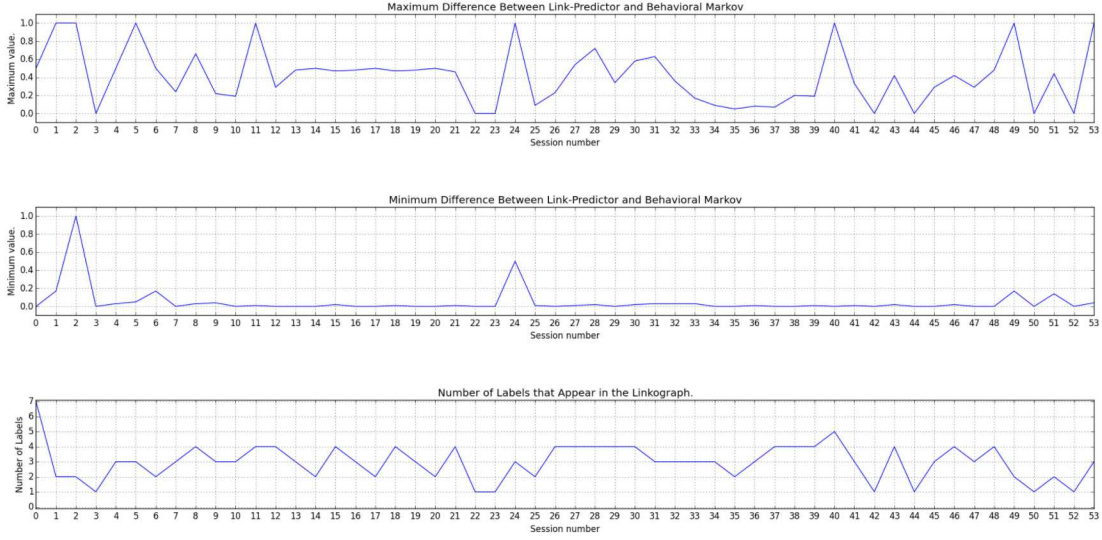
Here is the corresponding transition matrix for the behavioral model. The columns and rows are: Cleanup, Execute, For, Look, Move, Net_Start and Transfer.

$$\begin{bmatrix} 0.44 & 0.0 & 0.0 & 0.27 & 0.29 & 0.0 & 0.0 \\ 0.01 & 0.34 & 0.0 & 0.2 & 0.42 & 0.01 & 0.01 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.0 & 0.0 \\ 0.01 & 0.06 & 0.0 & 0.68 & 0.24 & 0.0 & 0.01 \\ 0.02 & 0.13 & 0.0 & 0.25 & 0.57 & 0.0 & 0.02 \\ 0.0 & 0.18 & 0.0 & 0.09 & 0.55 & 0.18 & 0.0 \\ 0.0 & 0.04 & 0.0 & 0.18 & 0.32 & 0.0 & 0.45 \end{bmatrix} \quad (\text{C.2})$$

Figure C.12. Behavioral transition matrix.

C.7 Link-Predictor Markov as a Next State Predictor

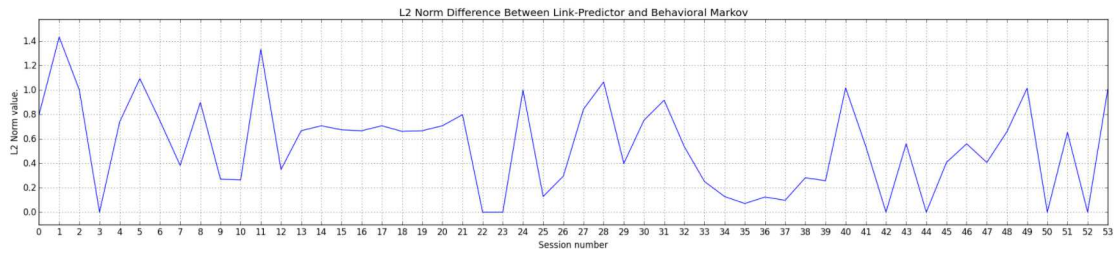
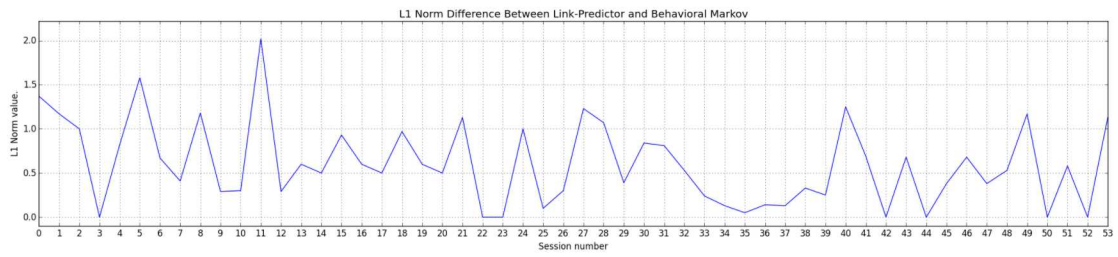
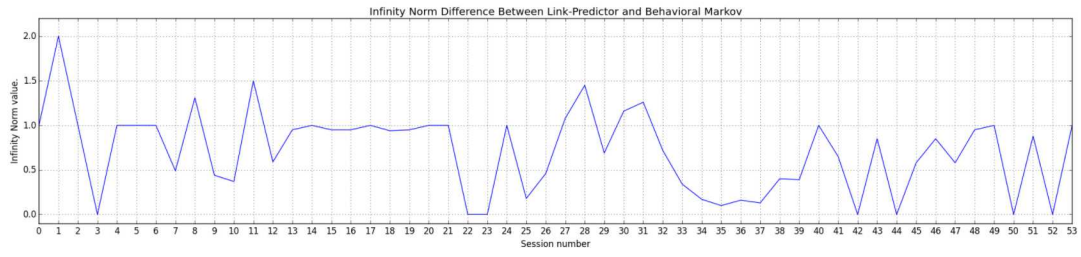
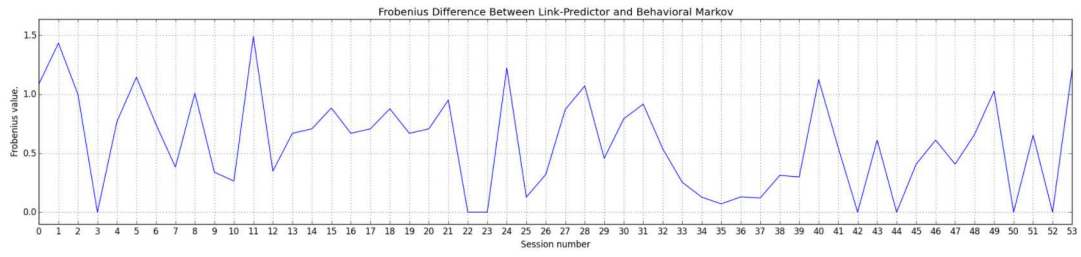
The following figure provides a measure of how well the Link-Predictor Markov models do in predicting the next state. The top figure gives the maximum entry-wise difference, in absolute value, between the link-predictor Markov models and the behavioral Markov models. The middle figure gives the minimum entry-wise difference, in absolute value, between the link-predictor Markov models and the behavioral Markov models. The last figure gives the number of states in the Markov model: that is, the number of abstraction classes that appear as a label in the linko-graph. The Markov models that use all the sessions are in these graphs as Session 0. Note that in both the top and middle figures, transitions are ignored that have a zero probability in both models.



From these graphs, we see that the maximum difference that occurs between transitions in the link-predictor model and the behavioral model can be high, for both a small number of abstraction class labels and a large number of abstraction class labels. For example, Session 5 has a maximum difference of 1.0 with only two labels and Session 46 has maximum difference of 1.0 with five labels. Note that since we are dealing with probabilities, 1.0 is the largest difference possible. On the other hand, the difference can be zero, for both a small number of abstraction class labels and a large number of abstraction class labels. For example, Session 2 has a minimum difference of 0.0 with only one label, and Session 46 has a minimum difference of 0.0 with five labels. Thus, we see that it is possible for the link-predictor Markov model to be a good indicator for some transitions and a poor indicator for others.

C.8 Distance Between Matrices

Another way to get an idea of how different the link-predictor Markov models are from the behavioral Markov models, we can consider the distance between the corresponding matrix representations given by some common norms. The next two figures show the distance between the link-predictor Markov model and the behavioral Markov model for each session. From top to bottom and left to right, the distances are with respect to the Frobenius norm, Infinity norm, L1 norm and L2 norm. Again Session 0 is the Markov model for all events.



This page intentionally left blank.

Appendix D

Appendix: Map Reduce Algorithms

Notation and assumptions:

- $O = (Abs, R)$ is an ontology with abstraction classes Abs and relations R .
- $F : \{0, 1, \dots, N-1\} \rightarrow Abs$ is a labeling.
- L_D is the derived linkograph using ontology O and labeling F .
- L_T is a target linkograph on N nodes.
- Optimal or best ontology is relative to the accuracy metric.

In determining the ontology O_F that gives the derived linkograph, L_D , that best approximates L_T , it is not necessary to calculate the accuracy every time a relation is considered for removal or addition. It is enough to calculate the overlinks and underlinks associated with each relation $A \rightarrow B$. In other words, the accuracy may change as relations are added or removed, but for a given relation $A \rightarrow B$, which accuracy is better between adding or removing $A \rightarrow B$ is the same independent of the other possible relations. Thus, each relation can be processed simultaneously to determine which should be added or removed. Moreover, the full accuracy does not need to be calculated, only the two sets $S_{(A,B)} =$ possible links (n, m) with $F(n) = A, F(m) = B$, and (n, m) in L_T and $Sb_{(A,B)} =$ possible links (n, m) with $F(n) = A, F(m) = B$ and (n, m) not in L_T . From these two sets, the overlinks and underlinks can be derived, given whether the relation $A \rightarrow B$ is in or not in the ontology. In fact, the determination of whether to add or remove $A \rightarrow B$ can be determined by checking the relative size of $S_{(A,B)}$ and $Sb_{(A,B)}$. If $S_{(A,B)}$ is larger, then add $A \rightarrow B$. If $Sb_{(A,B)}$ is larger, then remove $A \rightarrow B$.

Although it is enough to calculate $S_{(A,B)}$ and $Sb_{(A,B)}$, there is not really a savings in terms of complexity to replacing the accuracy calculation with finding these sets. The reason is that the cost of calculating the two sets is the same as the cost of calculating the accuracy; in both cases, every link needs to be considered. However, the runtime can be improved in two ways. The first is to replace the first accuracy calculation with finding the sets $S_{(A,B)}$ and $Sb_{(A,B)}$ and to replace the accuracy calculations in the loop with set cardinality comparison. This process requires one to only pay the accuracy cost once, which means that the cost of the algorithm in terms of the number of nodes N , reduces to a single cost at the beginning. This savings in runtime comes at a cost of

additional memory for saving the sets. The second method to improve runtime is to exploit the parallelism. In this direction, we consider a MapReduce [2] type methodology.

The idea is to split the links processed by a node according to the abstraction classes in the ontology and use these to form the records in the MapReduce methodology. Specifically, split the set of all possible links for L_T into the set of links (n, m) where n has a particular label under the labeling F . So, the records are $(A, *) := (n, m) \in M_N | L(n) = A$ where M_N is the maximal linkograph on N nodes, that is, the set of possible links. The Map function will then process these records and emit a count $((A, B), (s_{(A,B)}, sb_{(A,B)}))$ that provides a key (A, B) on the labels for a link and a count $(s_{(A,B)}, sb_{(A,B)})$ for the size of $S_{(A,B)}$ and $Sb_{(A,B)}$. The Reduce function handles summing $s_{(A,B)}$ and $sb_{(A,B)}$. With these sums, we can calculate the Boolean value $S > Sb$ which determines whether the relation $A \rightarrow B$ should or should not be in the ontology.

In the following algorithms: A indicates the abstraction class of the initial node, B indicates the abstraction class of the terminal node, F is the labeling, and O is the ontology.

D.1 Optimal Ontology Map

Algorithm 3 Optimal Ontology Map

```

1: function MAP( $A$ , linkograph,  $F$ , links)  $\triangleright$  links: the set of possible links in linkograph whose
   initial node is in  $A$ 
2:   for link in links do
3:     terminalNode  $\leftarrow$  terminal node of link
4:     if link in linkograph then
5:       EMIT( $((A, F(\text{terminalNode})), (1, 0))$ )
6:     else
7:       EMIT( $((A, F(\text{terminalNode})), (0, 1))$ )
8:     end if
9:   end for
10: end function

```

D.2 Optimal Ontology Reduce

Algorithm 4 Optimal Ontology Reduce

```

1: function REDUCE( $A, B$ , partialCountInLinkograph, partialCountNotInLinkograph)
2:   for all input  $((A, B), (\text{partialCountInLinkograph}, \text{partialCountNotInLinkograph}))$  do
3:      $s_{(A,B)} \mathrel{+}= \text{partialCountInLinkograph}$ 
4:      $sb_{(A,B)} \mathrel{+}= \text{partialCountNotInLinkograph}$ 
5:   end for
6:   EMIT( $((A, B), s_{(A,B)} > sb_{(A,B)})$ )
7: end function

```

D.3 High Impact First Minimum Similarity Reduce

Algorithm 5 High Impact First Minimum Similarity Reduce

```
1: function REDUCE(A, B, partialCountInLinkograph, partialCountNotInLinkograph, O)
2:   for all input ((A, B), (partialCountInLinkograph, partialCountNotInLinkograph)) do
3:      $S += \text{partialCountInLinkograph}$ 
4:      $Sb += \text{partialCountNotInLinkograph}$ 
5:   end for
6:   if  $S > Sb$  then
7:     if  $A \rightarrow B$  not in ontology then
8:        $\text{EMIT}((A, B), (0, s_{(A,B)} + sb_{(A,B)}))$ 
9:     else
10:       $\text{EMIT}((A, B), (1, s_{(A,B)} + sb_{(A,B)}))$ 
11:    end if
12:   else if  $S < Sb$  then
13:     if  $A \rightarrow B$  in O then
14:        $\text{EMIT}((A, B), (0, s_{(A,B)} + sb_{(A,B)}))$ 
15:     else
16:        $\text{EMIT}((A, B), (1, s_{(A,B)} + sb_{(A,B)}))$ 
17:     end if
18:   else
19:      $\text{EMIT}((A, B), (0, s_{(A,B)} + sb_{(A,B)}))$ 
20:   end if
21: end function
```

DISTRIBUTION:

- 1 MS 0359 D. Chavez, LDRD Office, 1911
- 1 MS 0899 Technical Library, 9536 (electronic copy)

