SAND2016-2597C

Linkography Ontology Refinement and Cybersecurity

Robert Mitchell, Andrew Fisher, Scott Watson, John Jarocki
Sandia National Laboratories *
Albuquerque, NM USA
{rrmitch, anfishe, srwatso, jcjaroc}@sandia.gov

ABSTRACT

The competition between cyber attackers and defenders is fundamentally a game. In this game, the stakes are high, the decisions are difficult and the timescale is very short. To date, most researchers in this area have focused on the strategic level decisions. This focus enables what-if scenarios that hinge on the opening move of the game. However, this approach does not allow for flexibility after the players choose these high-level opening moves. We compare this situation to a turn-based style of play where we hope to end the game quickly, for example, by halting the execution of a software program when we detect a signature that matches some definition of malicious.

With this research, we advance the field by addressing a finer granularity of move sequences. To this end, we propose techniques that scale to near real time, while processing micro-behavioral activity cues. We also propose a system that is automatable and that users can refine as their adversaries evolve. Previous work showed that linkography can analyze, provide models for and identify defensive opportunities at this fine level of granularity.

This article extends that work to narrow the remaining gap: the refinement of our models to support analysis of new or evolving adversaries. The result of this investigation, ontology refinement, applies to domains other than cyber security. In this way, we also hope to advance the state of the art for linkography in general.

1. INTRODUCTION

As cyber security researchers, we benefit from linkography because it allows us to examine the behavioral aspects of an adversary. We can consider a full spectrum of behavior granularity: from micro-behavior spanning seconds or minutes to macro-behavior spanning months or years. In our domain, listing directory contents and uploading malware are examples of micro-behavior; on the other hand, stealing personally identifiable information (PII) and infiltrating a supervisory control and data acquisition (SCADA) system are examples of macro-behavior. Our current focus with linkography is on micro-behavior, which is important to our discipline because it gives us the opportunity to apply countermeasures against cyber attackers in real time. This information, in turn, allows us to disrupt an attacker's higher level objectives, or macro-behavior. While prior work has established the usefulness of linkography in our and many other domains (for example, architecture, design, project management and social networking), creating linkographs has remained a mostly manual process that is ad hoc and error prone. To advance the state of the art of linkography, graph generation must be more scientific and repeatable.

Previously, Fisher, et al. [2] demonstrated a methodology to partially automate the construction of linkographs. By taking a set of abstraction classes and defining relations between these classes, a linkograph can be constructed algorithmically. However, this process still requires the correct set of abstraction classes and relations to be known a priori. This work helps to alleviate this problem by providing a system to evolve the relations between the classes as more information is obtained, thereby not requiring a priori knowledge of the definite relationship scheme.

2. BACKGROUND

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.

Gabriela Goldschmidt, creator of linkography, summarizes its history in her monograph [5]. 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. The author uses the term ontology to refer to the sets of labels that comprise these time series: for example, the function-behavior-structure (FBS) ontology. We use ontology to refer to a set of abstraction classes and a set of relations between these abstraction classes that transform a labeled time series into a linkograph.

Gero, et al. [4] propose a seven stage model of protocol analysis using linkography:

*Sandia National Laboratories is a multi-program 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 unlimited release: SAND2016-xxxx

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

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

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 article aims to fill this void by introducing methods for creating and refining ontologies.

Pourmohamadi and Gero [9] 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 [4] model. While Linkographer is specific to the FBS ontology, we propose a system that adapts to abstraction classes from any ontology.

Becattini, et al. [1] propose a framework based on OTSM-TRIZ [8] to automate protocol analysis. OTSM and TRIZ are transliterations from 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 [4] model. While their study is tailored for protocol analyses of designers, we have constructed a framework usable for any domain.

Kan and Gero discuss using cluster analysis, specifically the SPSS TwoStep cluster algorithm, to automatically identify clusters in a linkograph in [6] and extract other information from a linkograph in [7]. This addresses step seven (linkograph analysis) of Gero, et al.'s [4] model.

Fruchter [3] 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 [4] model. RECALL is applicable specifically to the design domain, while our investigation is germane to any activity.

Fisher, et al. [2] propose using linkography to characterize and disrupt cyber attackers. The authors identify the labeled time series to linkograph transform (Gero, et al.'s [4] sixth step) as an open question that we pursue in this 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. [2] 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. In this study, we propose methods to create and refine the linkograph ontologies.

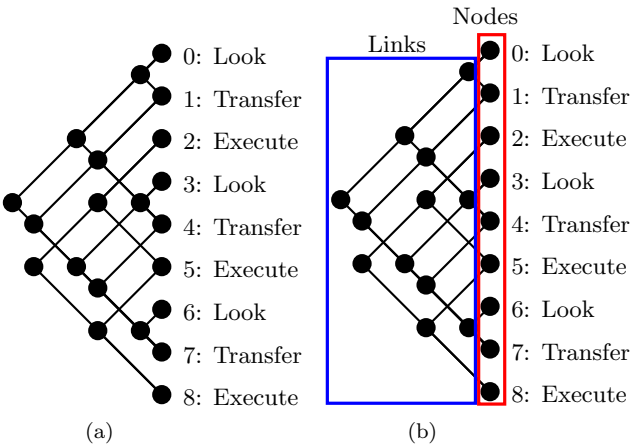

Figure 1: Example linkograph. (a) A linkograph on 9 nodes. (b) Linkograph with the nodes and links marked.

Our basic linkography process comprises four steps:

1. record a time series of interactions between the attacker and their defensive target
2. categorize those actions being taken
3. move to a different level of abstraction
4. create an annotated graphical object which shows relatedness

3. LINKOGRAPHY

This section provides the basic linkography definitions. Linkographs are temporal graph structures that consist of a sequence of events, a set of nodes labeled by these events and a set of links connecting the nodes. An example linkograph is shown in Figure 1. In this figure, the sequence of events is Look, Transfer, Execute, Look, Transfer, Execute, Look, Transfer and Execute. The nodes are the dots on the right and the links are any dots to the left, as depicted in Figure 1b. The nodes are referred to as node 0, node 1 and so on, where node 0 is the topmost node.

Intuitively, linkographs are meant to represent relationships between different events in time. The linkograph in Figure 1a indicates that the first Look event at node 0 is related to the subsequent Transfer events at nodes 1, 4 and 7. In general, the sequence of events can be anything; however, following the example of Fisher, et al. [2], this article considers a sequence of Windows console commands that are binned into abstraction classes. For example, the sequence of events in Figure 1a can be obtained from the commands in Listing 1 by assigning 'echo' commands to Look, 'scp' commands to Transfer and '.exe' commands to Execute.

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*

```
0: echo /etc/passwd >> /tmp/alef.txt
(Look)
1: scp /tmp/alef.txt subtle.c2.ca
(Transfer)
2: alpha.exe
(Execute)
3: echo /etc/group >> /tmp/bet.txt
(Look)
4: scp /tmp/bet.txt subtle.c2.ca
(Transfer)
5: beta.exe
(Execute)
6: echo /etc/resolv.conf >> /tmp/gimel.txt
(Look)
7: scp /tmp/gimel.txt subtle.c2.ca
(Transfer)
8: gamma.exe
(Execute)
```

Listing 1: A sequence of Windows console commands together with their associated abstraction labels.

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 $L : \mathbb{N}_{<N} \to Abs$, where $\mathbb{N}_{<N}$ is the set of natural numbers less than N and Abs is a set of strings called* abstraction classes.

With this definition, the linkograph in Figure 1 is given by

$$\mathcal{L} = \{(0,1),(0,4),(0,7),(1,4),(1,7),(2,5),\\ (2,8),(3,4),(3,7),(4,7),(5,8),(6,7)\}$$

together with the labeling, L, given by

$$\begin{aligned} &L(0) = \text{Look}, \quad L(1) = \text{Transfer}, \quad L(2) = \text{Execute},\\ &L(3) = \text{Look}, \quad L(4) = \text{Transfer}, \quad L(5) = \text{Execute}, \\ &L(6) = \text{Look}, \quad L(7) = \text{Transfer}, \quad L(8) = \text{Execute}. \end{aligned} \tag{1}$$

In general, the links in a linkograph, $\mathcal{L}$, are not required to respect a given labeling L, that is, it is possible to have $L(i) = L(i')$ and $L(j) = L(j')$ with $(i,j) \in \mathcal{L}$ and $(i',j') \notin \mathcal{L}$. An important subclass of linkographs is derived linkographs; for this subclass, a set of relations between the abstraction classes is used to create the links. The relations are given by an ontology, which is formally defined as follows:

Definition 2 (Ontology). *An ontology, $\mathcal{O}$, is a pair (Abs, R) where Abs is a set of abstraction class strings and $R \subset Abs \times Abs$ is the set of* relations *between the abstraction classes. The relation $(A,B) \in R$ is written $A \to B$. 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: $|\mathcal{O}| = |Abs|$.*

An example of an ontology is given in Figure 2. In this ontology, the Look abstraction class is related to the Transfer abstraction class, the Transfer abstraction class is related to itself and the Execute abstraction class is related to itself. Using these relations, a linkograph can be constructed from a sequence of abstraction classes by linking the nodes according to the relations in the ontology. For example, the relation Look → Transfer implies that every node labeled Look should be linked to every subsequent node labeled Transfer. The formal definition is as follows:

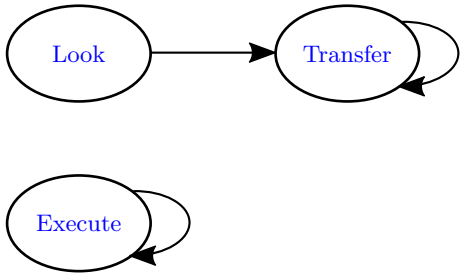

Figure 2: An example of an ontology.

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

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

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

The linkograph in Figure 1a is an example of a derived linkograph. In fact, it is the linkograph $\mathfrak{L}(\mathcal{O}, L)$, where $\mathcal{O}$ is the ontology given in Figure 2 and L is the labeling defined by Equation 1. See Fisher, et al. [2] for more details.

In derived linkographs, every time there is an event labeled A and a following event labeled B, either all such pairs are linked or none of them are linked. However, the human may see that some of the links should be present while others should not be present. This situation represents a case where the abstraction classes are too coarse to capture the relations that the human sees. In Section 4, we explore how to choose the relations in the ontology so the derived linkograph is as close to what the human sees as possible without changing the level of abstraction.

4. ONTOLOGY REFINEMENT

Traditionally in linkography, subject matter experts built linkographs using pen and paper. The way to advance the state of the art is to automate. We do not intend to remove the human from the control loop; we want to encourage humans to do what they are good at (abstract thinking) while offloading well defined, large scale, error prone tasks to a machine.

As indicated in Section 3, given a set of abstraction classes and an ontology, one can form a representative linkograph for a sequence of cyber-events like those in Listing 1; however, this requires one to already know the abstraction classes and the ontology before any events are received. Currently, the only method for producing the abstraction classes and ontologies is a trial-and-error, manual process. One solution to this dilemma is to introduce a systematic process of refinement whereby one determines how to update the system given new information.

In the context of cyber-events and linkography, a refinement process translates into a way to update the abstraction classes and ontology when given another cyber-event or sequence of cyber-events. With this process in place, one could start with a naive (for example, no relations, every relation or only the self-loops) ontology and allow the refinement process to improve it.

A simple ontology to start with is the self-loop ontology depicted in Figure 3. In general, a self-loop ontology is an

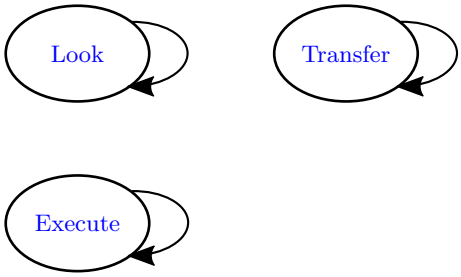

Figure 3: A self-loop ontology.

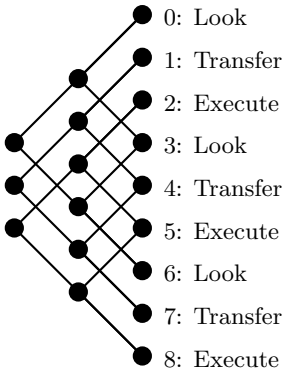

Figure 4: Derived linkograph using the ontology in Figure 3.

ontology that consists of just the self-loops for each of the abstraction classes. Figure 4 shows the derived linkograph when the self-loop ontology of Figure 3 is applied to the labeling in Equation 1. As expected, the linkograph relates Looks together, Transfers together and Executes together. However, is this the best we can do? Consider again the commands in Listing 1. The text file alef.txt appears in both the 0th and 1st command. Thus, these two commands are clearly related and should be linked. Similarly, the 3rd and 4th commands should be linked and 6th and 7th commands should be linked. Furthermore, the Look commands are on different files and are not related. Thus, the 0th, 3rd and 6th command should not be linked. Making these edits to Figure 4 produces the linkograph in Figure 5.

By changing the linkograph, the analyst provides more information on how the abstraction classes in the ontology

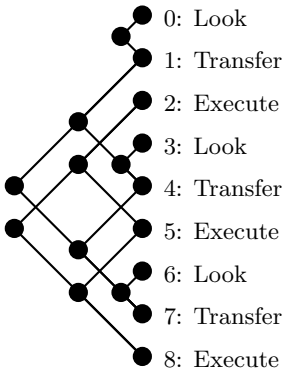

Figure 5: Linkograph in Figure 4 following human modification.

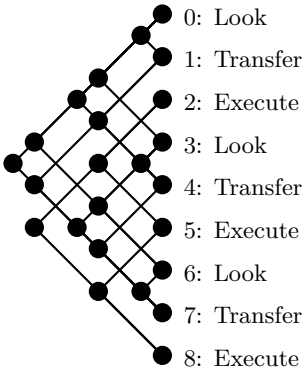

Figure 6: Derived linkograph using the ontology in Figure 3 with the relation Look → Transfer added.

should be connected. The goal is then to modify the ontology so that this new target linkograph is reproduced. But how can ontologies be modified? An ontology can be modified in four ways: adding additional relations, removing relations, adding abstraction classes or removing abstraction classes. For the method presented in this article, we restrict our attention to only adding or removing relations. In general, it may not be possible to add or remove a relation in such a way that the target linkograph is reproduced. For example, after adding the relation Look → Transfer to the ontology in Figure 3, the derived linkograph becomes the linkograph shown in Figure 6. Adding the Look → Transfer relation provides the desired links $(0, 1)$, $(3, 4)$ and $(6, 7)$; however, it also adds the undesired links $(0, 4)$, $(0, 7)$ and $(3, 7)$. Since removing the relation removes the desired links $(0, 1)$, $(3, 4)$ and $(6, 7)$, while adding the relation adds the undesired links $(0, 4)$, $(0, 7)$ and $(3, 7)$, neither removing nor adding the relation produces a derived linkograph that matches the target linkograph in Figure 5.

Since removing or adding relations does not guarantee an ontology that will produce a derived linkograph that is the same as a target linkograph, the next question is how to change the ontology so that the derived linkograph best approximates the target linkograph. The first step is to create a metric to measure the difference between the derived linkograph and the target linkograph. The basic idea behind the metric that we use is to count the number of relations that are present in exactly one of the two linkographs. To start the definition, let the linkograph $\mathcal{L}$ on N nodes be the target linkograph, $L : \mathbb{N}_{<N} \to Abs$ be a labeling, $\mathcal{O}$ be an ontology and $\mathcal{L}'$ be the derived linkograph for the labeling L and ontology $\mathcal{O}$. Furthermore, let $A, B \in Abs$ be two abstraction classes. As seen above, adding a relation $A \to B$ to an ontology can cause the derived linkograph $\mathcal{L}'$ to have links between nodes, with initial node labeled A and terminal node labeled B, that are not present in the target linkograph $\mathcal{L}$. Let $O(A, B)$ be the set of these links and call these links *overlinks*. Formally, define

$$O(A, B) = \{(n, m) \in \mathcal{M}_N \mid L(n) = A \wedge L(m) = B \wedge (n, m) \in \mathcal{L}' \wedge (n, m) \notin \mathcal{L}\}.$$

Note that since the derived linkograph contains all links (n, m) with $L(n) = A$ and $L(m) = B$, it is not possible for such a link (n, m) to be in $\mathcal{L}$ and not in $\mathcal{L}'$. Thus, when

$A \to B$ is in the ontology, then $O(A, B)$ contains all the links with initial node labeled A and terminal node labeled B that are different between the linkographs $\mathcal{L}$ and $\mathcal{L}'$. On the other hand, when $A \to B$ is not in $\mathcal{O}$, then $\mathcal{L}'$ does not contain any of the links (n, m) with initial node labeled A and terminal node labeled B. Thus, $O(A, B) = \emptyset$. The dual notion to consider, in this case, is that of *underlinks*, which are the links present in $\mathcal{L}$ that are not present in $\mathcal{L}'$. Formally, define

$$U(A,B) = \{(n,m) \in \mathcal{M}_N \mid L(n) = A \wedge L(m) = B \\ \wedge (n,m) \notin \mathcal{L}' \\ \wedge (n,m) \in \mathcal{L}\}.$$

Dual to overlinks, when $A \to B$ is not in $\mathcal{O}$, then $U(A, B)$ contains all the links (n, m) with $L(n) = A$ and $L(m) = B$ where $\mathcal{L}'$ and $\mathcal{L}$ differ, and when $A \to B$ is in $\mathcal{O}$, $U(A, B) = \emptyset$.

By taking the union of $O(A, B)$ and $U(A, B)$, we get all links with initial node labeled A and terminal node labeled B where the derived linkograph and target linkograph differ. Performing this union for all pairs of abstraction classes $A, B \in Abs$, yields $\mathcal{L}\Delta\mathcal{L}' := \bigcup_{A,B \in Abs} \{O(A, B) \cup U(A, B)\}$, which contains all links that are different between $\mathcal{L}$ and $\mathcal{L}'$. Thus, the size of the set $\mathcal{L}\Delta\mathcal{L}'$, denoted $|\mathcal{L}\Delta\mathcal{L}'|$, can be used to measure the difference between the two linkographs. If $|\mathcal{L}\Delta\mathcal{L}'|$ is normalized by the total possible links, $|\mathcal{M}_N|$, and subtracted from 1, we get a measure of the difference between the two linkographs that is 0 if they share nothing in common and 1 if they are the same. Define the accuracy of $\mathcal{L}'$ with respect to $\mathcal{L}$ as

$$\mathcal{A}(\mathcal{L}, \mathcal{L}') = 1 - \frac{|\mathcal{L}\Delta\mathcal{L}'|}{|\mathcal{M}_N|}.$$

Returning to the example of the linkograph $\mathcal{L}$ in Figure 5 and the current ontology $\mathcal{O}$ in Figure 3, the accuracy $\mathcal{A}(\mathcal{L}, \mathcal{L}')$ can be used to determine if any relation $A \to B$ should be added to or removed from the ontology $\mathcal{O}$. As a concrete example, consider the relation Look $\to$ Look. With this relation, the accuracy between the derived and target linkographs is $\mathcal{A}(\mathcal{L}, \mathcal{L}') = 0.83$. However, if the relation Look $\to$ Look is removed to produce the derived linkograph $\mathcal{L}''$, then the accuracy is $\mathcal{A}(\mathcal{L}, \mathcal{L}'') = 0.92$. So, the relation should be removed from the ontology.

Now that we have a method to determine if a relation should be added or removed, the next step is to iterate the process to achieve progressively better approximations. As an initial approach, we consider all the possible relations to add to and all the possible relations to remove from the ontology and add or remove the relation that leads to the best improvement in the accuracy. Then, repeat the process a given number of times. Algorithm 1 illustrates this method that we call the Brute Force Minimum Similarity (BFMS) algorithm. This algorithm is recursive and starts with an initial ontology o and a target linkograph lg. The variables o' and a' keep track of the current best ontology and resulting accuracy from the target linkograph, respectively. Similarly, o_0 and a_0 keep track of the change in the ontology and accuracy for a single relation added, while o_1 and a_1 keep track of the change in the ontology and accuracy for a single relation removed. Finally, the variable m limits the number of times the ontology is modified.

With any algorithm, it is usual to consider the complexity in terms of important input variation. The complexity is important for comparing the BFMS algorithm with a more refined version that we introduce later. We consider the complexity dependent on the number of nodes, N, in the linkograph, the size of the set of abstraction classes, $|Abs|$ and the number of allowable changes, m. To start the analysis, we describe complexity estimates for each of the subfunctions involved in the algorithm, starting with `caclulateaccuracy`. This function has a worst case complexity of $O(\frac{N(N-1)}{2}) = O(N^2)$, since a basic algorithm for calculating this quantity considers every possible link in $\mathcal{M}_N$ and counts those links that are exactly in one of the two linkographs. The size of the set $\mathcal{M}_N$ is $\frac{N(N-1)}{2}$ since it is the number of pairs of possible nodes [2]. Finally, the rest of the accuracy calculation is just the $O(1)$ operations of subtraction and division. To analyze the complexity of operations involving the ontology, we assume that the ontology is stored as a set of abstraction classes Abs and a set of relations R. Furthermore, for a worst case upper bound, we assume the number of relations is $|Abs|^2$, which is the maximum possible number of relations for $|Abs|$ abstraction classes. Under these assumptions, copying the ontology (`copy.deepcopy`), is an $O(|Abs| + |Abs|^2) = O(|Abs|^2)$ operation, subtracting a relation (`subtractrelation`) is an $O(1)$ operation, and adding a relation (`addrelation`) is an $O(1)$ operation. Lastly, the function `createabstractionclasses` considers the abstraction class label on every node, which takes $O(N)$ operations. For a worst case bound, we assume that the resulting set of

Algorithm 1 Brute Force Minimum Similarity

```
a' ← calculateaccuracy(o, lg)
o' ← copy.deepcopy(o)
if 0 < m then
    abstractionclasses ← createabstractionclasses(lg)
    for initial in abstractionclasses do
        for terminal in abstractionclasses do
            pair ← (initial, terminal)
            o0 ← copy.deepcopy(o)
            if initial in o then
                if terminal in o[initial] then
                    o0 ← subtractrelation(o0, pair)
                else
                    o0 ← addrelation(o0, pair)
                end if
            else
                o0 ← addrelation(o0, pair)
            end if
            a0 ← calculateaccuracy(o0, lg)
            o1, a1 ← recurse(lg, o0, m - 1)
            if a0 > a' then
                o' ← o0
                a' ← a0
            end if
            if a1 > a' then
                o' ← o1
                a' ← a1
            end if
        end for
    end for
end if
return o', a'
```

abstraction classes, `abstractionclasses`, is the full set Abs.

Now that we have estimates for each of the sub-functions, we can derive an estimate for the complete BFMS algorithm. To start, we let $P(|Abs|, N, m)$ be the cost of the algorithm. Note that $P(|Abs|, N, 0) = O(1)$, since line 3 exits the algorithm when $m = 0$. As mentioned above, lines 1, 2 and 3 contribute a cost of $O(N^2)$, $O(|Abs|^2)$, and $O(N)$, respectively. Collectively, their cost reduces to $O(N^2 + |Abs|^2)$, provided $N > 1$. The loops starting at lines 5 and 6 contribute $O(|Abs|^2 \cdot L(|Abs|, N, m))$ where $L(|Abs|, N, m)$ is the cost of the interior of the loops given by lines 7-27. Letting the cost of all interior $O(1)$ operations be c, the cost of $L(|Abs|, N, m)$ is $O(|Abs|^2 + N^2 + P(|Abs|, N, m-1) + c)$ due to the copying of the ontology on line 8, the accuracy calculation on line 18 and the recursive call on line 19. Thus, we have an estimate for $P(|Abs|, N, m)$ as:

$$P(|Abs|, N, m) = O(|Abs|^2 \cdot (|Abs|^2 + N^2 + P(|Abs|, N, m-1)))$$

To simplify the unraveling, we consider three cases achieved by holding two of the three $|Abs|$, N and m constant while varying the third. The estimates are then

$$Abs, N \text{ fixed} \Rightarrow O(|Abs|^{2(m+1)}),$$
$$Abs, m \text{ fixed} \Rightarrow O(N^2),$$
$$N, m \text{ fixed} \Rightarrow O(|Abs|^{2(m+1)}).$$

In other words, the BFMS algorithm is exponential in m, quadratic in N and polynomial in $|Abs|$.

One obvious question to ask is whether or not we can choose m large enough so that the accuracy is maximized. This question leads, in turn, to a subsequent question of whether or not there is an ontology that maximizes the accuracy in the first place. The answer is yes there is an ontology that maximizes $\mathcal{A}(\mathcal{L}', \mathcal{L})$. To see there is a best ontology, it is important to note that adding or removing a relation $A \to B$ only affects the links involving an initial node labeled A and a terminal node labeled B. Thus, one can consider the accuracy when the relation $A \to B$ is present and when it is removed. Each relation needs to be considered only once. This idea is formalized in Theorem 1. Before stating the theorem, it is convenient to introduce another pair of sets $S(A, B)$ and $\bar{S}(A, B)$ related to links between abstraction classes A and B. The set $S(A, B)$ is the set of links from nodes labeled A to nodes labeled B that are in $\mathcal{L}$ and $\bar{S}$ are those links that are not in $\mathcal{L}$. Formally, we define

$$S(A, B) = \{(n, m) \in \mathcal{M}_N \mid L(n) = A \wedge L(m) = B \wedge (n, m) \in \mathcal{L}\}.$$

and

$$\bar{S}(A, B) = \{(n, m) \in \mathcal{M}_N \mid L(n) = A \wedge L(m) = B \wedge (n, m) \notin \mathcal{L}\}$$

Using these two sets, statements about overlinks and underlinks can be transferred to statements about sets that do not refer to the derived linkograph. As noted above, when $A \to B \in \mathcal{O}$, then $U(A, B) = \emptyset$ and $O(A, B)$ contains all the possible links from nodes with initial label A to nodes with terminal label B that are different between $\mathcal{L}'$ and $\mathcal{L}$. Moreover, $\bar{S}(A, B) = O(A, B)$, in this case, since the only difference between the two definitions is that $O(A, B)$ requires the link (n, m) to be $\mathcal{L}'$, which is automatically satisfied when $A \to B \in \mathcal{O}$. Thus, $\bar{S}(A, B)$ contains all the possible links whose initial nodes are labeled A and whose terminal nodes are labeled B that are different between $\mathcal{L}'$ and $\mathcal{L}$. A similar analysis leads to $S(A, B)$ containing all differing links whose initial nodes are labeled A and terminal nodes are labeled B when $A \to B \notin \mathcal{O}$. It follows that minimizing the links that differ between $\mathcal{L}$ and $\mathcal{L}'$ amounts to determining which is the smaller set: $\bar{S}(A, B)$ or $S(A, B)$. If $S(A, B)$ is smaller, then do not include $A \to B$ in $\mathcal{O}$, so then, the two linkographs differ by $S(A, B)$. On the other hand, if $\bar{S}(A, B)$ is the smaller set, then include $A \to B$ in $\mathcal{O}$, so then, the two linkograph differ by $\bar{S}(A, B)$. By using this procedure, $\mathcal{L}\Delta\mathcal{L}'$ is made as small as possible, which in turn, makes $\mathcal{A}(\mathcal{L}, \mathcal{L}')$ as large as possible. Thus, we have proven:

THEOREM 1. *Let $\mathcal{L}$ be a linkograph on N nodes, $\mathcal{O}$ be an ontology with abstraction classes Abs, $L : \mathbb{N}_{<N} \to Abs$ be a labeling with labels in Abs and $\mathcal{L}' = \mathfrak{L}(\mathcal{O}, L)$ be the derived linkograph. The accuracy $\mathcal{A}(\mathcal{L}, \mathcal{L}')$ is maximized by the ontology $\mathcal{O}$ if and only if $\mathcal{O}$ satisfies the following two conditions:*

- *if $|\bar{S}(A, B)| > |S(A, B)|$, then $A \to B \in \mathcal{O}$,*
- *if $|\bar{S}(A, B)| < |S(A, B)|$, then $A \to B \notin \mathcal{O}$.*

From Theorem 1, we see that to find the ontology that produces a derived linkograph whose accuracy compared to the target linkograph is maximized, it is enough to consider every relation $A \to B$ once. Thus, we can do better than the BFMS algorithm by removing the recursive call and just looping through each possible relation. We still include a limiting number m of changes that are allowed so that the new ontology is not too different from the starting ontology. Furthermore, we sort the relations according to which relations have the potential to impact the accuracy the most. This potential is measured by determining, for each relation $A \to B$, how many pairs of nodes have the initial node labeled A and terminal node labeled B. The resulting algorithm, which we call High Impact First Minimum Similarity (HIFMS), is shown in Algorithm 2. In Algorithm 2: o is the original ontology, lg is the target linkograph, o' is the current best ontology, a' is the associated current best accuracy, $\hat{o}$ is the proposed ontology, $\hat{a}$ is the associated proposed accuracy and m is the maximum allowed changes.

To analyze the complexity of the HIFMS algorithm, we make the same assumptions as before, that is, we let N be the number of nodes in the linkograph, we let $|Abs|$ be the number of abstraction classes, and we assume that the ontology is implemented as a set of abstraction classes and a set of relations. We then get the same estimate of $O(\frac{N(N-1)}{2} + (|Abs'| + |Abs|^2) + N) = O(N^2 + |Abs|^2)$ for lines 1-4. Again we assume that `abstractionclasses` is the full set Abs. Then, creating the `pairlist` in lines 6-10 is an $O(|Abs|^2)$ operation. For sorting the pairs in line 11, we will assume a Mergesort complexity bound, so the worst case complexity for line 11 is given by $O(|Abs|^2 \log(|Abs|^2))$ which simplifies to $O(|Abs|^2 \log(|Abs|))$. For the loop starting at line 12, the interior has a complexity on the order of $O(|Abs|^2 + \frac{N(N-1)}{2} + c)$ for some constant c representing the number of $O(1)$ operations. This estimate reduces to $O(|Abs|^2 + N^2)$. But how many times does this loop run? Unlike in the case of the BFMS algorithm, where the maximum number of changes m effectively causes the loop to

Algorithm 2 High Impact First Minimum Similarity

```
1: a' ← calculateaccuracy(o, lg)
2: o' ← copy.deepcopy(o)
3: if 0 < m then
4:     abstractionclasses ← createabstractionclasses(lg)
5:     pairlist ← []
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 ← sortpairlist(pairlist, lg)
12:    for pair in sortedpairlist do
13:        ô ← copy.deepcopy(o')
14:        if pair[0] in ô then
15:            if pair[1] in ô[pair[0]] then
16:                ô ← subtractrelation(ô, pair)
17:            else:
18:                ô ← addrelation(ô, pair)
19:            end if
20:        else:
21:            ô ← addrelation(ô, pair)
22:        end if
23:        â ← calculateaccuracy(ô, lg)
24:        if â > a' then
25:            m -= 1
26:            a' ← â
27:            o' ← ô
28:            if 0 == m then
29:                break
30:            end if
31:        end if
32:    end for
33: end if
34: return o', a'
```

repeat, the `break` condition of lines 28 and 29 provides an early termination for the loop contingent on m. Thus, the loop may not execute for every pair in `pairlist`. However, if m is large enough, for example, if m is larger than size of `pairlist`, then the loop will execute for every pair in `pairlist`. So, as a worst case bound, we assume that the loop executes $|Abs|^2$ times, which is the size of `pairlist`. Thus, as a worst case bound, we have

$$O(N^2 + 2|Abs|^2 + |Abs|^2 \log(|Abs|) + |Abs|^2 * (|Abs|^2 + N^2))$$

If we fix $|Abs|$ and consider the complexity in N, then we get an estimate of $O(N^2)$. If we fix N and consider the complexity in $|Abs|$, we get an estimate of $O(|Abs|^4)$. Thus, the algorithm is quadratic in the number of nodes and polynomial in the number of abstraction classes. In practice, the number of abstraction classes will not be large, so the complexity bound in $|Abs|$ should not be large. For example, in Figures 2 and 3 the number of abstraction classes is $|Abs| = 3$. However, the algorithm can be adapted to change the ontology in place, which will knock the complexity in $|Abs|$ down to quadratic.

5. IMPLEMENTATION

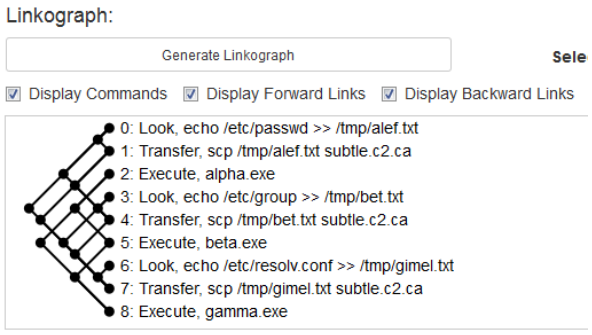

Figure 7: LinkShop-derived linkograph from Figure 2 ontology.

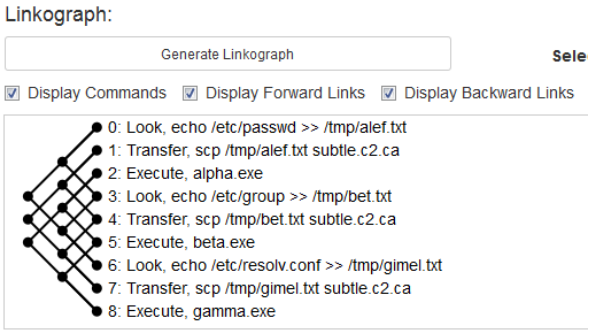

Figure 8: LinkShop-derived linkograph from Figure 3 ontology.

The LinkShop system is a multi-purpose tool for visualizing and manipulating the models detailed in Fisher, et al. [2]. The system provides a full pipeline for linkograph analysis via features such as saving and displaying models, modifying abstractions and ontologies visually, creating linkographs and generating and comparing linkograph and sub-linkograph statistics. Extending the system to support ontology refinement required adding the ability for a researcher to add or remove links from an existing linkograph. When performing an ontology refinement the researcher first selects an ontology and a linkograph. Figures 7 and 8 show the researcher experimenting with the Figure 2 and 3 ontologies, respectively. Next, the researcher modifies the linkograph by adding and removing desired relations as shown in Figure 9 and selects the maximum number of allowable changes. The system then performs the HIFMS algorithm on the inputs and returns the resulting ontology as well as the calculated accuracy. The results are then displayed to the researcher as shown in Figure 10.

6. NUMERICAL RESULTS

In our pursuit of improving the rigor of linkography work, we searched for key metrics. We converged on accuracy and similarity, which we discussed in Section 4. Accuracy measures the fidelity with which an ontology can reproduce a given linkograph. Recall that accuracy is given by:

$$\mathcal{A}(\mathcal{L}, \mathcal{L}') = 1 - \frac{|\mathcal{L} \Delta \mathcal{L}'|}{|\mathcal{M}_N|} \quad (2)$$

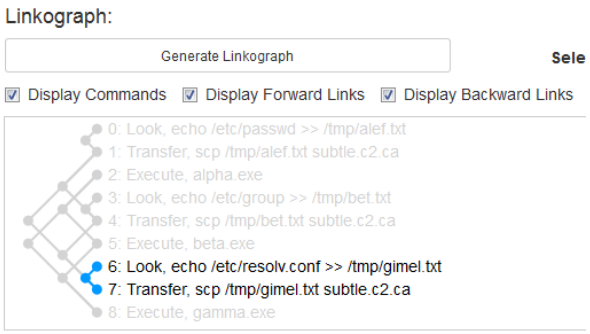

Figure 9: Human-modified linkograph in LinkShop.

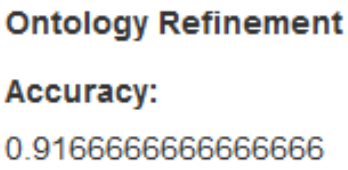

Figure 10: Accuracy report of refined ontology in LinkShop.

Similarity is the number of relation modifications allowed to transform between two given ontologies. Table 1 presents the accuracy performance of HIFMS given different minimum similarities. As a basis of comparison, we use an ontology with the maximum accuracy obtained by considering all possible relations. Our experimental input is a data set of 47 console sessions. With the maximum constraint, at most one change, HIFMS yields an overall accuracy of 75.2%. As expected, when we relax this constraint, the accuracy improves and approaches the theoretical limit of 81.2%.

Any practical ontology refinement algorithm should have an implementation with a reasonable computational complexity. In particular, we are concerned about time complexity. Our time complexity metric is very simple: run time in milliseconds (ms). Tables 2 and 3 present the runtime performance of BFMS and HIFMS. While the time complexity of BFMS grows exponentially with the maximum changes allowed, the time complexity of HIFMS is near constant, which matches our algorithmic complexity estimations in Section 4 since the number of nodes and abstraction classes remain constant. The 19 ms penalty (31 ms versus 12 ms) observed when maximum changes is equal to one is explained by HIFMS calculating impactfulness and sorting the relations at the beginning.

7. CONCLUSIONS AND FUTURE WORK

In this article, we have demonstrated the construction of both linkographs and ontologies for generating them in order to analyze cyber attacks. We have shown how the links

Table 1: Accuracy of HIFMS.

HIFMS (1)	HIFMS (2)	HIFMS (3)	HIFMS (4)	Threshold (Ideal)
75.2%	79.0%	80.2%	80.5%	81.2%

Table 2: Time complexity of BFMS.

Maximum Changes	Solutions Considered	Runtime (ms)
1	25	12 ms
2	$25 + 25^2$	190 ms
3	$25 + 25^2 + 25^3$	4882 ms
4	$25 + 25^2 + 25^3 + 25^4$	124847 ms
5	$25 + 25^2 + 25^3 + 25^4 + 25^5$	3192626 ms

Table 3: Time complexity of HIFMS.

Maximum Changes	Solutions Considered	Runtime (ms)
1	< 25	31 ms
2	< 25	31 ms
3	< 25	32 ms
4	< 25	33 ms
5	< 25	36 ms

in a derived linkograph stem from an ontology, and more importantly, how an improved ontology results from human modifications to a linkograph. This process of ontology refinement not only makes it possible to increase the automation of linkograph analysis, it provides a method to introduce minor adjustments that can improve the accuracy of an adversary model or track the evolution of an adversary adopting new tactics, techniques and protocols.

We have described our journey in the discovery of two ontology refinement algorithms: BFMS and HIFMS, which allow us to provide a definition for refinement accuracy based on the difference between the derived and target linkographs. This metric provides a way to systematically and continuously check progress. Our complexity analysis also enables quantitative measurement of the cost of achieving that accuracy.

By enhancing LinkShop to support the ontology refinement process using these algorithms, we can now demonstrate the effectiveness of this iterative process and provide a proving ground for researchers as well as a situational awareness tool for analysts.

Several opportunities exist for future work in this area. One avenue is to pursue a refinement methodology that incorporates changing the abstraction classes in conjunction with the relations; this addresses the inconsistent linking problem we discussed in Section 3. A more broad research direction is to consider models of behavior. Since the methods described in this article show how to refine ontologies and measure linkograph accuracy, this process can be used to quantitatively compare models, say, of two different attackers or the same attacker over time. Once cyber attack behavior has been modeled, we can model the defensive side at the same level of granularity. With both sets of models in hand, game theoretic methods can be used to create what-if scenarios. We could derive and measure the effectiveness of defensive strategies given a specific attacker model and provide feedback in real time to the defender when a strategy appears to be losing its edge. Finally, although we have designed the ontology refinement process in the context of linkography, we hope it sparks ideas for machine learning researchers who constantly run into the problem of a shortage

of reliable labeled data by introducing the idea of human-guided refinement of roughly characterized initial data.

8. REFERENCES

[1] N. Becattini, G. Cascini, and F. Rotini. An OTSM-TRIZ Based Framework Towards the Computer-Aided Identification of Cognitive Processes in Design Protocols. In J. S. Gero and S. Hanna, editors, *Design Computing and Cognition*, pages 99–117. Springer International Publishing, 2015.

[2] A. Fisher, K. Carson, D. Zage, and J. Jarocki. Using Linkography to Understand Cyberattacks. In *IEEE Conference on Communications and Nework Security*, Florence, Italy, September 2015.

[3] R. Fruchter. Bricks & bits & interaction. In *New Frontiers in Artificial Intelligence*, pages 35–42. Springer, 2001.

[4] J. S. Gero, J. W. Kan, and M. 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.

[5] G. Goldschmidt. *Linkography: Unfolding the Design Process*. The MIT Press, 2014.

[6] J. W. Kan and J. S. Gero. Design behaviour measurement by quantifying linkography in protocol studies of designing. *Human behaviour in designing*, 5:47–58, 2005.

[7] J. W. Kan and J. S. Gero. Acquiring information from linkography in protocol studies of designing. *Design Studies*, 29:315–337, 2008.

[8] N. Khomenko and D. Kucharavy. OTSM-TRIZ problem solving process: solutions and their classification. In *Proceedings of TRIZ Future Conference*, pages 6–8, Strasbourg, France, November 2002.

[9] M. Pourmohamadi and J. S. Gero. LINKOgrapher: An Analysis Tool to Study Design Protocols Based on FBS Coding Scheme. In *Proceedings of International Conference on Engineering Design*, Copenhagen, Denmark, August 2011.