NETWORK ANALYSIS WITH STOCHASTIC GRAMMARS

DISSERTATION

Alan C. Lin, Maj, USAF

AFIT-ENG-DS-15-S-014

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-15-S-014

NETWORK ANALYSIS WITH STOCHASTIC GRAMMARS

DISSERTATION

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Alan C. Lin, BS, MS

Maj, USAF

September 2015

AFIT-ENG-DS-15-S-014

NETWORK ANALYSIS WITH STOCHASTIC GRAMMARS

Alan C. Lin, BS, MS
Maj, USAF

Committee Membership:

Gilbert L. Peterson, PhD.
Chair

Robert F. Mills, PhD.
Member

Michael R. Grimaila, PhD, CISM, CISSP
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

AFIT-ENG-DS-15-S-014

Abstract

Digital forensics requires significant manual effort to identify items of evidentiary interest from the ever-increasing volume of data in modern computing systems. One of the tasks digital forensic examiners conduct is mentally extracting and constructing insights from unstructured sequences of events. This research assists examiners with the association and individualization analysis processes that make up this task with the development of a Stochastic Context-Free Grammars (SCFG) knowledge representation for digital forensics analysis of computer network traffic.

SCFG is leveraged to provide context to the low-level data collected as evidence and to build behavior profiles. Upon discovering patterns, the analyst can begin the association or individualization process to answer criminal investigative questions.

Three contributions resulted from this research. First, domain characteristics suitable for SCFG representation were identified and a step-by-step approach to adapt SCFG to novel domains was developed. Second, a novel iterative graph-based method of identifying similarities in context-free grammars was developed to compare behavior patterns represented as grammars. Finally, the SCFG capabilities were demonstrated in performing association and individualization in reducing the suspect pool and reducing the volume of evidence to examine in a computer network traffic analysis use case.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Peterson, for his guidance and support throughout the course of this dissertation effort. The insight and experience was certainly appreciated.

Alan C. Lin

Table of Contents

List of Figures

List of Tables

List of Algorithms

NETWORK ANALYSIS WITH STOCHASTIC GRAMMARS

I. Introduction

1.1 General Issue

Digital forensics involves identifying and analyzing relevant fragments of computer data to piece together a probable explanation, or narrative, of events that transpired, for investigative and judiciary purposes [1]. The ever-increasing volume of digital data collected for forensic examinations increases the difficulty of this task. Law enforcement agencies find themselves unable to commit the human resources necessary to manually sift through the data, which results in the examination bottlenecking the overall criminal investigative process [2–4].

This research focuses on speeding examination by automating portions of the computer network traffic analysis. Computer network traffic evidence is usually in the form of a network packet capture in one or more packet capture (PCAP) files. Examining PCAP files is time consuming because the packet-by-packet format of the data captured within PCAP files is not reflective of how a typical computer end-user thinks and operates the system—the user performs a specific task and leaves the computer to carry out the underlying mechanics required to accomplish the user action. It is often the actions of evidentiary interest that the examiner is attempting to identify. Examiners need tools and technologies are necessary to help examiners "efficiently identify the relevant pieces of data in a timely manner [5, 6]."

Development of forensic tools and technologies require an understanding of the legal system since legal experts, not computer scientists or network administrators, conduct judicial proceedings. Digital evidence must be presented in understandable language for legal and non-technical persons to use in court cases [7]. In an ideal scenario, a prosecutor first defines the legal question for the forensic examiner, who then decides on a scientific method to extract the relevant evidence necessary. This coordination makes the most efficient use of resources and also provides an end-state for the examination [8].

Evidence examination, digital and physical, must satisfy both legal and scientific requirements [9]. To bridge the gap between the legal requirements and the forensic examination procedures, Inman and Rudin [10] propose a framework of four forensic processes intended to answer the investigative questions, "who, what, when, where, why, and how." While their framework was originally developed for physical evidence, Pollitt [8] discusses how each of these are adaptable in digital forensics as well.

1.1.1 Identification.

The identification process attempts to answer the "what" question. In physical evidence, identification uses a set of characteristics or features to determine the classification or category of an item [9]. For instance, upon recovering a bullet, identification may use its size to determine that "the bullet is a 9mm bullet." Identification works similarly in digital forensics, where features, such as protocol and headers, may reveal the type of transmission or the type of file [8].

1.1.2 Individualization.

Individualization takes the identification process further, by attempting to make or use uniqueness assertions to answer "which one" or "whose is it" questions [10]. For instance, identification may type a recovered bullet as a 9mm, individualization attempts to singularly identify that “this is a 9mm from a specific gun.” Individualization in digital forensics might be found in header information that includes the user-agent and username [11].

1.1.3 Association.

Association makes inference about the origin of source or the likelihood that two items were in contact [10]. Extending the previous bullet examples, an association would connect the 9mm bullet to the victim or shooter. In digital forensics, association is identifying evidence connecting the suspect or victim [9], such as modus operandi patterns or demonstrations of the commission of the crime [8].

1.1.4 Reconstruction.

Reconstruction attempts to answer “where” and “when” questions [10]. In physical evidence, this phase is often last in the examination process because it requires elements of prior processes [9]. Using the previous bullet example again, reconstruction is relevant that a bullet embedded into a wall after hitting a person only after first identifying the bullet as a 9mm, individualizing it to the suspect’s gun, and associating it as the bullet that passed through the victim. Time stamps are common in digital media, making reconstruction more accessible, though the examiner still needs to account for the possibility of tampering and synchronization [8]. In both physical and digital settings, reconstruction tends to focus on the relative order rather than the specific timing of events [10].

1.2 Hypothesis

This research hypothesizes that stochastic context-free grammar (SCFG) parsing and structure inference techniques can generate activity patterns and discover patterns in computer network traffic to distinguish between routine and irregular activities. The goal is to eliminate suspects or data to focus the digital forensic examination and extract and contextually structure the raw data into higher-level information necessary for criminal investigators to answer investigative questions [12].

1.3 Contributions

Three contributions resulted from this research. The first contribution is a step-by-step approach to apply SCFG to novel domains. The approach is a result of a cross-domain examination that identified domain characteristics that result in a positive application of the SCFG knowledge representation. The second contribution led to the development of a grammar-comparison method resulting from the association process experimental setup. Finally, performing association and individualization process demonstrated SCFG mechanics to reduce the suspect pool and the volume of evidence in computer network traffic examinations.

1.3.1 General Methodology to Apply SCFG to a Novel Domain.

To evaluate SCFG as a knowledge representation for computer network traffic analysis, we survey problem domains that leverage SCFG, such as natural language processing [13–15], activity recognition [16–19], bioinformatics [20–24], and automated planning [13, 25–27]. The survey highlights suitable domain traits for SCFG representation and its suitability for networking data, resulting in the development of a methodology to adapt SCFG to new domains, which can assist other researchers in applying SCFG towards their problem domains.

1.3.2 Grammar Comparison.

After determining suitability for computer network traffic analysis, this research examines representing profiles as behavioral patterns and associating generated activity sequences back to the originating profile. The association process requires a set of known profiles. Conducting the association process first required a grammar-based comparison methodology to evaluate the differences between the profiles. The need to illustrate the differences in the profiles led to the development of a graph-based grammar comparison measure with polynomial computation complexity. The profile comparison showed which profiles shared common causal symbol patterns as a measure of similarity.

Generically, grammar comparison enables comparison of data patterns, rather than the individual data elements; this has applications in language translation and other infinitely large domains, where full enumeration of the complete set of outputs is not possible [28]. Grammar comparison also has implications in the ability to identify incompatibilities in parsing code between different compilers [29].

1.3.3 Association and Individualization with SCFG

Finally, we investigate using SCFG for association and individualization tasks. The grammar comparison contribution enabled the association experimental setup, which applied SCFG parsing on an unknown activity sequence by a profile to produce a parse likelihood value. The parse likelihood value is demonstrated as an effective means of identifying the originating profile, thereby eliminating suspects that do not fit the profile.

In individualization, structure inference to discover behavior patterns to reduce the amount of activities in a computer network capture, while retaining the anomalous activities of interest. Discovered behavior patterns are unique characteristics inferred from the individualization process. Instead of using the patterns to make the individualization assertion, the patterns are used to “provide investigative leads” by identifying events that are not attributable to patterns [8]. Using SCFG for pattern discovery led to the development of two grammar inference methods to find behavior patterns exhibited by the individual. Because grammar inference is not domain specific, the developed inference approaches are applicable to other domains using a SCFG knowledge representation.

1.4 Methodology

Stochastic Context Free Grammar (SCFG) is a hierarchal, rule-based, knowledge representation. Because of the discrete symbols representing atomic events and causal rules enforced by the production rules, they are applicable to abstracting the transition between distinct events, focusing on the relative sequence of events without accounting for the specific duration of each activity. The application of SCFG to computer network traffic analysis required the development of procedures and algorithms, shown in Figure 1 flowchart as rectangular process boxes.

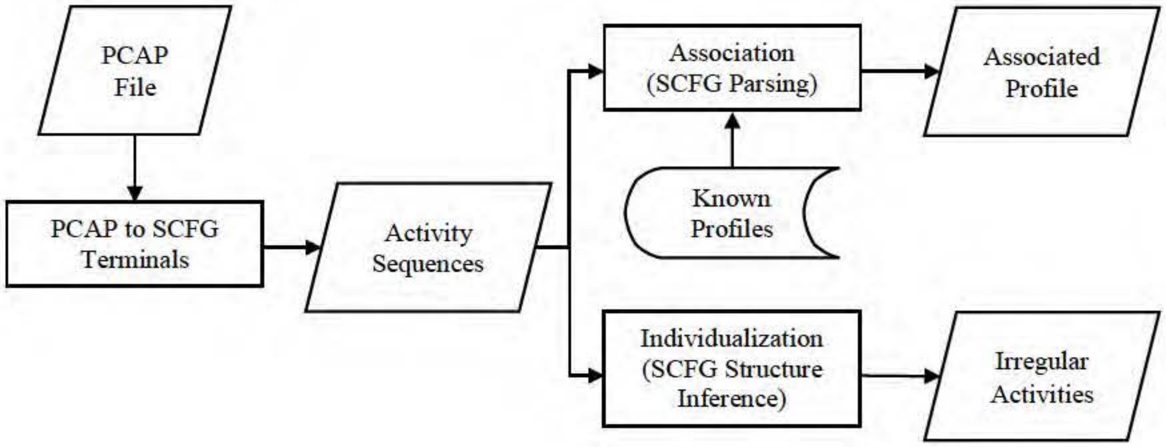


Figure 1. Methodology Process Overview.

1.4.1 PCAP to SCFG Terminals

The PCAP to SCFG Terminals process required a survey of other domains that apply SCFG knowledge representation. The survey involved a cross comparison of the characteristics of the data from each of the domains, the applications of each domain, and adaptations required to enable SCFG representation. After reaching the conclusion that computer network traffic data is suitable for SCFG representation, we performed the PCAP to SCFG Terminal process using a PCAP file from a self-contained digital forensic scenario [11] and generated an activity sequence of SCFG terminals.

1.4.2 Association

Performing the association process requires activity sequences from the PCAP to SCFG Terminal process and a set of behavioral profiles. Association applies SCFG parsing to identify the originating source, using parse likelihood as a quantitative measure. This process requires a set of known profiles in SCFG form and examining this process required a comparison measure of the profile themselves. Investigation into grammar-based methods to evaluate output-to-output [30, 31] or rule-to-rule comparisons [32, 33] methods revealed that they were either undecidable [33–35] or beyond polynomial computational complexity [32]. Identifying similarities between grammars and graphs, we developed grammar comparison methodology based on an iterative graph node-matching algorithm [36], which enabled comparison of grammars by comparisons of their symbol causalities. This comparison was performed on the profile grammars to verify desired similarities and differences.

With understanding of the differences between the profiles, the association process parses an unknown activity sequence with each profile, generating a quantitative measure from total and most-probable parse likelihood to compare between the different profiles. In this manner, SCFG parsing associates each sequence to the profile based on that yielded the greatest parse likelihood.

1.4.3 Individualization

To reduce the amount of data under examination, the individualization process uses alignment and bigram-based SCFG structure inference learning to collectively discover routine behavior patterns, which isolate activities that could not be attributed to the discovered patterns. Conducting this process on a network capture over three sessions identified the known anomalous event. Eliminating the events explainable by routine behaviors reduced the data size without reducing the anomalous event.

1.5 Structure

The remainder of this dissertation is as follows:

- Chapter II provides background on SCFG knowledge representation, the notation and provides the rationale behind using it for network processing. In addition, this chapter presents the general methodology for applying SCFG on a novel domain. The discussion includes SCFG adaptations to better represent certain domain characteristics.
- Chapter III describes SCFG in the network forensic applications and presents the algorithms used in each. We present and use a graph-based methodology for grammar comparison and the algorithms used in the alignment and bigram inference structure learners.
- Chapter IV provides the experimental setup for the methodology outlined in the previous chapter and analysis of the results.
- Chapter V summarizes SCFG usage on computer network traffic and identifies future work.
- Appendix A provides additional background and related work on grammar comparison, including applications for such methods.
- Appendix B provides background and related work on SCFG for test data generation, as a potential future application of using SCFG to generate computer network traffic that mimic patterned user behavior.

II. Stochastic Context-Free Grammars

Stochastic Context-Free Grammar (SCFG) is a hierarchal, rule-based, knowledge representation capable of expressing a variety of domains. This chapter presents SCFG fundamentals to facilitate understanding how SCFG are leveraged and how SCFG enables reasoning on the represented domain. The application to computer network forensics proposed in the next chapter use SCFG parsing and structure, described in this chapter, to automate portions of the association and individualization tasks.

This chapter begins by presenting SCFG definition and notation. Next, the chapter identifies several problem domains and their respective SCFG applications. These examples serve to assist in understanding the notations and concepts. Then, a subsection discusses algorithms and identifies readily available implementations for parsing with SCFGs; one application of computer network traffic analysis uses parsing with SCFGs. Finally, this chapter presents a methodology to apply SCFG onto novel domains, including potential domain adaptations.

2.1 Stochastic Context-Free Grammar (SCFG): Definition and Notation

Knowledge representation functions as a surrogate for an actual idea or concept that enables "pragmatically efficient computation [37]." As a means to represent domain knowledge, Stochastic Context-Free Grammars (SCFGs) use rules to describe the order between symbols that represent different concepts depending on the problem domain. SCFG provides a structural order to enable contextual understanding of low-level data.

An SCFG is written as a 4-tuple, $G = \langle V_T, V_N, P, S\rangle$, where:

- V_T is the finite set of terminal symbols. Terminal symbols are the lowest level observation in the domain and represent atomic, irreducible elements of the domain.

- V_N is the finite set of non-terminal symbols. Each non-terminal symbol is defined by a production rule. Each production rule and respective non-terminal reflects a specific combination of terminal observations. The non-terminal therefore represents a higher level concept than the low-level terminal symbols.

- P is the finite set of production rules. In SCFG, a production rule, r, is in the form, $A \rightarrow \gamma_1..\gamma_n\,[\delta]$, where $A \in V_N$, and $\gamma_i \in (V_T \cup V_N)$. Each r has a likelihood parameter, δ, where $0 \leq \delta \leq 1$ and the sum of all δ's of all r's with the same A must sum to 1. The "→" in the production rule notation means equivalence, meaning the sequence of symbols on the right-hand side is representable as the singular non-terminal symbol on the left-hand side. Conversely, the left-hand side symbol is representable as the sequence on the right-hand side. The operation that uses production rules equivalencies is called, *substitution*.

- S is the starting non-terminal symbol ($S \in V_N$). The purpose of identifying the starting symbol is its use in *parsing,* which is a grammar operation performed to read or generate output with the grammar. Production rules defining S are at the highest level of the SCFG rule hierarchy.

2.2 Stochastic Context-Free Grammar Methods

There are two mechanisms in which grammar reflects the domain knowledge: parsing and structure inference. Parsing is the process of applying a grammar to an observance and uses substitution operations and stochastic parameters to explain observances. Structure inference uses pattern discovery to produce a grammar structure from observances so that the grammar structure reflects patterns and covers all observances.

2.2.1 Parsing.

Top-down parsing generates a sentence from the starting symbol, while bottom-up parsing compresses the sentence into starting symbol. Figure 2 is an NLP part-of-speech example modified from [38] of an SCFG production rule set that reads a limited set of English sentences for the purpose of determining grammatical validity and meaning through part-of-speech assignment. In the NLP domain, each word is in V_T and the part-of-speech is in V_N. Each row is a production rule and the entire list of rules is *P*. Rules that define *S* are typically listed at the top, as in the figure.

V_T:	saw, man, woman, telescope, dog, the, with, in				
V_N:	S, VP, NP, PP, Vt, NN, DT, IN				
P:	S	→	NP	VP	[1.0]
	VP	→	Vt	NP	[0.8]
	VP	→	VP	PP	[0.2]
	NP	→	DT	NN	[0.7]
	NP	→	NP	PP	[0.3]
	PP	→	IN	NP	[1.0]
	Vt	→	saw		[1.0]
	NN	→	man		[0.1]
	NN	→	woman		[0.1]
	NN	→	telescope		[0.3]
	NN	→	dog		[0.5]
	DT	→	the		[1.0]
	IN	→	with		[0.6]
	IN	→	in		[0.4]
S:	S				

Figure 2. Example grammar [38] from a Natural Language Processing (NLP) domain SCFG.

The grammar has a starting symbol *S,* which means that all sentences from this grammar compresses into a proper noun (`NP`) followed by a verb (`VP`). Figure 3 shows a parse tree that indicates the different production rule substitutions for the sentence, "`the man saw the dog with the telescope,`" showing that this sentence can be parsed and understood by the grammar. In contrast, no series of production rule substitutions exist can compresses “`man in telescope`” into `S`. The grammar cannot parse any sentences irreducible to `S`.

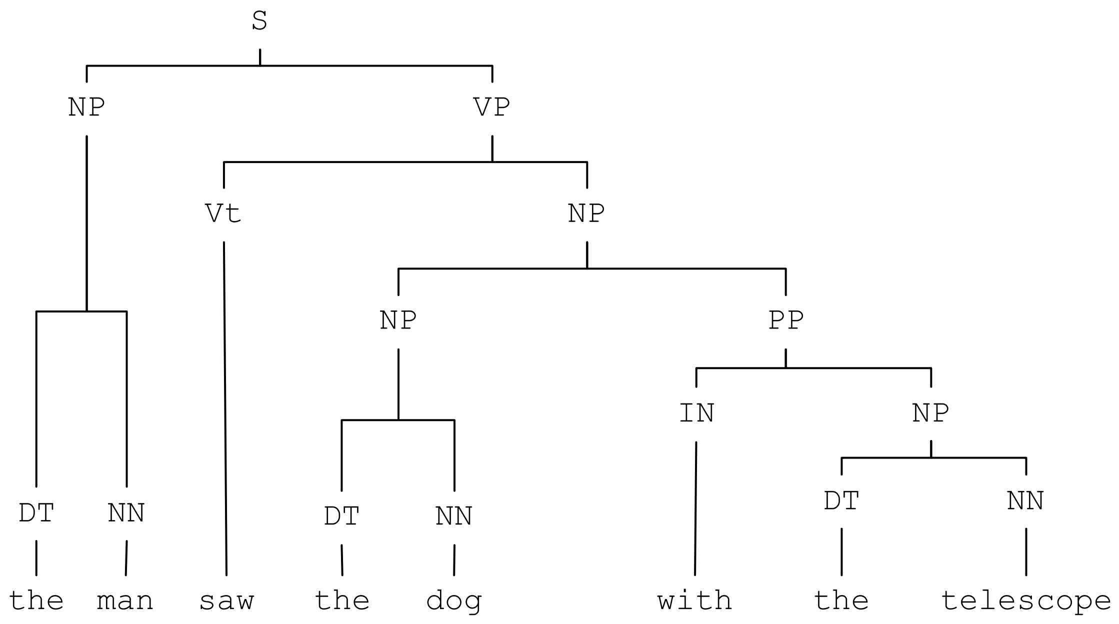


Figure 3. A parse tree of the sentence, “the man saw the dog with the telescope” using the grammar in Figure 2. Each terminal English word is substituted with a non-terminal symbol based on the available production rules until the sentence reduces to the starting terminal.

Figure 4 is a Blackjack grammar from the activity recognition domain [17]. The terminals are card and chip manipulations involved in playing a hand of blackjack. The non-terminals correspond to various phases of the game. Through substitution and production rule selection, the authors make inferences on the observed play by parsing the sequence using the grammar.

In the course of a game, the player must implement a strategy, denoted by the non-terminal symbol, `G`. Every legal blackjack game requires a substitution to `G`. The authors infer that the player is using a basic strategy if the parse of the play uses the production rule, `G → J → ff`. If a parse uses the other `G` substitutions, then the player is using more advanced strategies of “splitting pairs” or “doubling down.” The stochastic parameters also indicate that players are more likely to use a basic strategy, rather than an advanced strategy. In parsing sequences of plays from a single player, high incidences of advanced strategies may indicate a more advanced level player.

The previous example only applies the structural part of the production rules to make a determination whether a sentence was or was not from a given grammar. Applying stochastic parameters for each production rule provide additional domain representation and inferencing capability [39].

The context-free characteristic of SCFG allows unconstrained production rule substitutions. This creates situations where there are multiple valid parses for a sentence. Stochastic parameters provide a quantitative means to disambiguate different interpretations. There are two ways to infer meaning from the previous example sentence, “`the man saw the dog with the telescope`.” The first meaning is that the man saw a dog next to a telescope, where the word “with” takes on the semantic meaning of “next to.” The sentence can also be read to mean that the man saw a dog using a telescope. Figure 5 shows the parse derivations for each of these meanings. Parse likelihood is calculated as the product of each of the production rule likelihoods, shown in the brackets, used in the derivation.

V_T: a – dealer removed card from house
b – dealer removed card from player
c – player removed card from house
d – player removed card from player
e – dealer added card to house
f – dealer dealt card to player
g – player added card to house
h – player added card to player
i – dealer removed chip
j – player removed chip
k – dealer pays player chip
l – player bets chip

V_N: S,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O

P:	S	→	AB	[1.0]	Blackjack
	A	→	CD	[1.0]	play game
	B	→	EF	[1.0]	determine winner
	C	→	HI	[1.0]	setup game
	D	→	GK	[1.0]	implement strategy
	E	→	LKM	[0.6]	evaluate strategy
		→	LM	[0.4]	
	F	→	NO	[0.5]	clean-up
		→	ON	[0.5]	
	G	→	**J**	**[0.8]**	**player strategy -**
		→	**Hf**	**[0.1]**	**Adv - Splitting**
		→	**bfffH**	**[0.1]**	**Adv - Doubling**
	H	→	l	[0.5]	place bets
		→	lH	[0.5]	
	I	→	ffI	[0.5]	deal card pairs
		→	ee	[0.5]	
	J	→	f	[0.8]	basic strategy
		→	fJ	[0.2]	
	K	→	e	[0.6]	house hits
		→	eK	[0.4]	
	L	→	ae	[1.0]	dealer downcard
	M	→	dh	[1.0]	player downcard
	N	→	k	[0.16]	settle bet
		→	kN	[0.16]	
		→	j	[0.16]	
		→	jN	[0.16]	
		→	i	[0.16]	
		→	iN	[0.18]	
	O	→	a	[0.25]	recover card
		→	aO	[0.25]	
		→	b	[0.25]	
		→	bO	[0.25]	

S: S

Figure 4. Blackjack Grammar [17]

Using the grammar in Figure 2, the top parse has a greater parse likelihood, where the top-parse is 0.000741 (1.0 × 0.7 × 1.0 × 0.1 × 0.8 × 1.0 × 0.3 × 0.7 × 0.5 × 1.0 × 0.6 × 0.7 × 1.0 × 0.3) likelihood versus the bottom-parse with 0.000494 (1.0 × 0.7 × 1.0 × 0.1 × 0.2 × 0.8 × 1.0 × 0.7 × 0.5 × 1.0 × 0.6 × 0.7 × 1.0 × 0.3) likelihood. Based on the grammar likelihoods, the interpretation top interpretation is more likely than the bottom interpretation.

In instances where a sentence has multiple parses, equations (1) and (2) outline the difference between the *parse likelihood* and the *most-probable parse likelihood*, where $p(t)$ is the parse likelihood for a single parse tree and n is the number of valid parses.

$$Parse\ Likelihood = \sum_{1}^{n} p(t) \tag{1}$$

$$Most\ Probable\ Parse\ Likelihood = \max\, [p(t)]_1^n \tag{2}$$

2.2.1 SCFG Parsing Algorithms.

Parsing and probabilistic parameters enable inference from an SCFG. Both have pragmatic, small constant polynomial computational complexity for practical applications.

There are multiple SCFG parsing algorithms. For conciseness, this subsection discusses only the Cock-Younger-Kasami (CYK) algorithm [40] and the Earley

Algorithm [41]. Both algorithms have a small integer exponential, $O(n^3)$ time complexity and an $O(n^2)$ space complexity. The CYK algorithm approaches parsing using a bottom-up approach by using a table to store incrementally longer substitutions of the sentence and tracking valid combinations of production rule used for substitutions until the sentence reaches S.

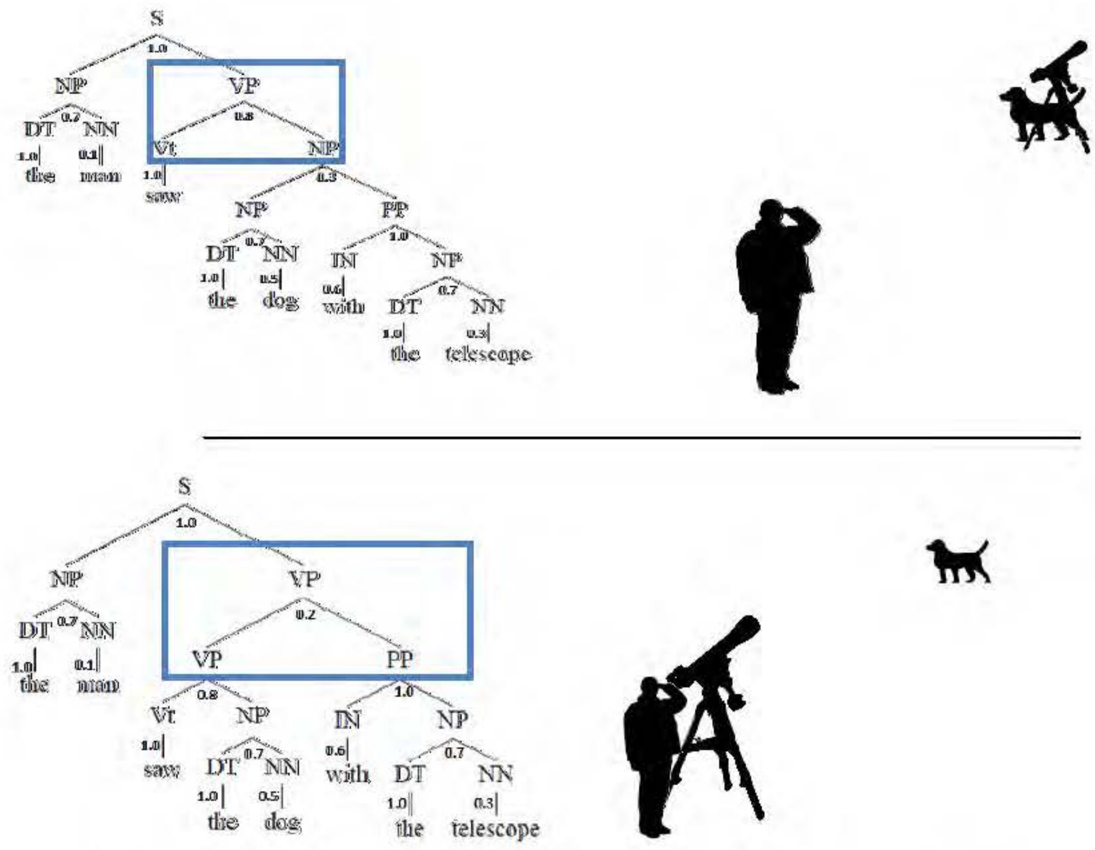


Figure 5. Two parse tree derivations for "the man saw the dog with the telescope [38]".

In contrast, the Earley Algorithm uses dynamic programming to track possible partial parsing states starting from S in a top-down fashion until substitutions match the sentence; the Earley Algorithm uses a prediction, scanning, and completion operator that determines checks or adds a state onto the list of parse states.

While it is possible to implement each parser using the pseudocode from [40, 41], parser implementations are publically available. The Stanford Natural Language Processing Group has a Java implementation[1], though the parser was designed primarily for the NLP domain. Royal Holloway University of London developed The Grammar Tool Box (GTB)[2] which more readily accepts domain-independent grammars. This researched extended an Earley parser written in C++ [3].

2.3 SCFG Application Domains

SCFG has sufficient expressiveness as a knowledge representation for domains such as natural language processing [13–15, 42–44], bioinformatics [20–24, 45, 46], activity recognition [16–19, 47], and automated planning [13, 25–27, 48, 49]. These domains use the inferencing capabilities to solve domain specific tasks. Some domains adapt the data or SCFG to better characterize the domain or make inferences that are more suitable. We examine the domains here for the purpose of identifying domain characteristics that are suitable for SCFG representation and potential adaptations available when applying SCFG on a novel domain.

2.3.1 Natural Language Processing.

Natural language processing (NLP) uses machine learning to understand and process human languages. NLP applications include document translation, user

[1] The Stanford Natural Language Processing Group, http://nlp.stanford.edu/software/index.shtml
[2] Royal Holloway University of London Grammar Tool Box (GTB), http://www.cs.rhul.ac.uk/research/languages/projects/gtb/gtb.html
[3] https://github.com/shaobohou/pearley

interfaces, speech recognition, and text processing [50]. SCFG knowledge representation enables language analysis, which involves a decomposition of a sentence through several stages. This decomposition typically requires several stages that stratify sentences into syntax (pattern structure), semantics (meaning), and pragmatics (contextual intent) [14]. Each word in the language is a terminal because the words are the lowest useful observable. The non-terminals represent parts of speech, and the production rules define valid parts of speech sequences.

Part-of-speech (POS) tagging is a common processing step that checks syntax and performs aspects of the semantics stage. The SCFG in Figure 2 is an example of an NLP grammar. Parsing reveals whether or not the words in the sentence follow a grammatically accepted order, defined by the grammar production rules. For instance, the example grammar has production rules a noun precedes a verb (S -> NP VP) and a determiner precedes only nouns (NP -> DT NN). A parsable sentence passes the syntax check.

To obtain more information from the knowledge representation, SCFG uses the POS tag in the parse to provide semantic meaning for each word. This aspect is necessary to resolve word ambiguity, where one word can take different meanings. SCFG enables the correct POS assignment, by assigning a POS that is valid in context of the adjacent words that follow the syntactic rules. With the correct POS, it would be possible to discern the meaning behind words like “can” which can take both noun (a container) and verb (to be able to) interpretation.

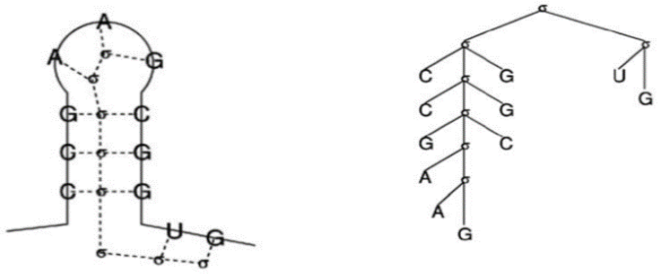
Beyond understanding the specific meaning being individual words, the SCFG parse likelihood disambiguates between multiple valid parses. Figure 3 shows two parses of the same sentence with the same POS tag for each word. The difference at the pragmatic level comes from the ordering of the production rules in each parse. The ordering chunks the sentences into parts that affects the interpretation of the sentence. In general, NLP applications use the production probabilities to infer the most probable parse by accepting the highest probability parse as the most correct interpretation.

2.3.2 Bioinformatics.

Bioinformatics is the application of computational techniques to analyze biological data [51]. Bioinformatics leverages SCFG in studying biological sequences such as DNA, RNA, and proteins [22]. Researchers in the field found that linguistic methods were applicable to biological sequences by capturing informational and structural aspects of macro-molecules [21]. The observables in sequences are limited to the amino acids that make up the sequence. For instance, the terminal set for RNA comprises of four nucleotides: adenine (A), cytosine (C), guanine (G), or uracil (U). The non-terminals correspond to substructures of these nucleotides. An example RNA structure and respective grammar is shown in Figure 6 [22].

SCFG was found sufficiently expressive to describe the variability in biological sequences, such as non-regular features in secondary structure of RNA [21, 52]. The symbolic, syntactic, semantic, and pragmatics stages of NLP are analogous to the sequence, structure, function, and role progression in biology [21]. Similar to parsing a set of text, parsing sequences with a grammar identifies sequences that from the same family. Other bioinformatics SCFG tasks include discriminating sequences between transfer RNA (tRNA) and non-tRNA, ascertaining secondary structure in new sequences, and finding common sequences present in a family of sequences [21, 52]. SCFG can also generate new sequences by applying top-down substitution. Dowell and Eddy [53] found SCFG grammars that nearly equaled the predictive power of the conventional physics-based energy minimization approach.

(C (C (GAAGC) G) G) UG



V_T: A,C,G,U

V_N: S,X_1,…,X_{16}

P:

$S \rightarrow A\ X_1\ U$	$X_2 \rightarrow X_3\ X_4$	$X_3 \rightarrow A\ X_5\ U$
$X_5 \rightarrow A\ X_6\ U$	$X_6 \rightarrow X_7\ X_8$	$X_7 \rightarrow A\ X_9\ U$
$X_9 \rightarrow G\ X_{10}\ C$	$X_{10} \rightarrow A\ X_{11}$	$X_{11} \rightarrow U\ G$
$X_8 \rightarrow G\ X_{12}\ C$	$X_{12} \rightarrow A\ X_{13}\ U$	$X_{13} \rightarrow A\ X_{14}$
$X_{14} \rightarrow G\ C$	$X_4 \rightarrow G\ X_{15}\ C$	$X_{15} \rightarrow C\ X_{16}\ G$
$X_{16} \rightarrow U\ G$		

Figure 6. A sample RNA secondary structure with its corresponding SCFG [22].

2.3.3 *Activity Recognition.*

Activity recognition is the analysis of sensor data to automatically detect recorded events of interest in surveillance or smart home applications [54]. The applications use SCFG to recognize complex events from combinations of simple or atomic actions recorded from one or more sensors [54, 55]. In contrast to NLP and bioinformatics, activity recognition does not have a natural or common basis for an SCFG terminal set. Terminal definition in activity recognition usually requires preprocessing step to identify discrete events, particular from sensors that take a continuous reading. The terminal set is often dependent on the number of sensors and the type of atomic actions each sensor can discern. The non-terminals then describe ordered combinations of events of interest specific to the application.

For example, Ivanov and Bobick [16] performed experiments at a gesture-level, where they attempted to infer the type of music from parsing sequences of atomic hand movements. This requirement only required one sensor to track the hand movement. They also conducted a single source video surveillance experiment to detect the activities of persons in a parking lot with a vehicle. Figure 7 is a partial grammar from their parking lot experiment. With this grammar, they used the grammar to infer the observed activity based on the actions of the car and the person. The ordering of rules made it possible to distinguish whether a person was entering the car or parking the car. The “ | ” symbol denotes a logical “OR” to group all productions with the same LHS symbol together.

Moore and Essa [17] extended the work in [16] and performed interaction-level experiment on blackjack games in which they assessed player behavior using the probabilistic parameters to profile whether the player was a novice or expert player and whether the play was a low or high-risk player based on observed strategies and betting amounts. Their blackjack SCFG is shown in Figure 4.

```
       TRACK  →     CAR-TRACK                                      [0.5]
                 |  PERSON-TRACK                                   [0.5]
   CAR-TRACK  →     CAR-THROUGH                                    [0.25]
                 |  CAR-PICKUP                                     [0.25]
                 |  CAR-OUT                                        [0.25]
                 |  CAR-DROP                                       [0.25]
  CAR-PICKUP  →     ENTER-CAR-B CAR-STOP PERSON-LOST B-CAR-EXIT    [1.0]
 ENTER-CAR-B  →     CAR-ENTER                                      [0.5]
                 |  CAR-ENTER CAR-HIDDEN                           [0.5]
  CAR-HIDDEN  →     CAR-LOST CAR-FOUND                             [0.5]
                    CAR-LOST CAR-FOUND CAR-HIDDEN                  [0.5]
  B-CAR-EXIT  →     CAR-EXIT                                       [0.5]
                 |  CAR-HIDDEN CAR-EXIT                            [0.5]
    CAR-EXIT  →     car-exit                                       [0.7]
                 |  SKIP car-exit                                  [0.3]
    CAR-LOST  →     car-lost                                       [0.7]
                 |  SKIP car-lost                                  [0.3]
    CAR-STOP  →     car-stop                                       [0.7]
                 |  SKIP car-stop                                  [0.3]
 PERSON-LOST  →     person-lost                                    [0.7]
                 |  SKIP person-lost                               [0.3]
```

Figure 7. SCFG production rules (partial) used in Ivanov and Bobick's [16] parking lot experiment.

2.3.4 Automated Planning.

Zimmerman and Kambhampati [56] define planning as achieving goals by constructing a sequence of actions based on the belief that actions have specific consequences. A plan is an observed action pattern or sequence [57] and analogous to a sentence in the NLP domain. Automated planning leverages SCFG in two ways. By using top-down substitutions, a grammar identifies possible actions to meet the top-level goal, defined by start terminal productions [49]. A plan is complete once the substitution reaches a list of terminals. If given a partial plan, the problem then attempts to recommend actions that complete the plan by making it parseable [56, 58].

Alternatively, a bottom-up parse of a plan reveals whether or not a plan satisfies the constraints of the goal [25]. An unparseable plan means that the goal is unreachable, based on the constraints placed by the production rules.

Similar to activity recognition, automated planning requires an application specific definition of the terminal set that define atomic, discrete actions. The non-terminals represent sub-activities, where the sub-activities are defined as sequences of lower level activities. A parse of a complete plan therefore reveals an hierarchy of how lower level goals accomplish higher level ones. The planning domain refers to this hierarchy as an hierarchical task network (HTN).

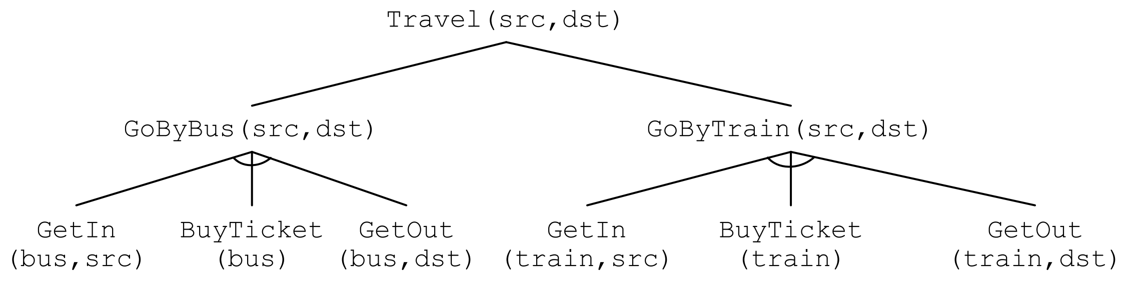
Geib and Steedman [13] outlined the parallels between NLP and plan recognition by translating a HTN into a CFG grammar. Plan operators define the effects of primitive atomic actions which are converted into a grammar's terminal symbols. Plan methods define non-primitive actions and convert into grammar productions, where the name of the method is a non-terminal on the left-hand side, and the sub-tasks are written as the production's right-hand side. This conversion assumes a totally ordered method definition, represented through the serial listing in the production's right-hand side. Their example, shown in Figure 8, translates an HTN method into a production [13].

(m1, acquire(shoes),
  {goto(store),choose(shoes),buy(shoes)},
  {(1 < 2), (2 < 3)})

acquire(shoes) → goto(store),choose(shoes),buy(shoes)

Figure 8. A plan recognition method translated into a grammar production [13].

Li, et al. [48, 59] leverage the probabilistic likelihood aspect of SCFG to ensure that the knowledge representation captures the preferences of the users. The production rules reflect the different sub-actions that the user takes and the production rule likelihood reflects the tendency for that user to perform that action, similar to the way that Moore and Essa [17] determined the complexity of a Blackjack player's strategy, based on his likelihood to split pairs or double down. The SCFG infers preferences from these tendencies to perform certain tasks or strategies over others. Li, et al. [27, 48] used a travel domain to illustrate how it is possible to infer a person's preferred mode of travel, as shown in Figure 9. Combined with learning, a planning system leverages the user's preference and biases to better assist the user in achieve goals [56].



Primitives (V_T): BuyTicket, GetIn, GetOut

Tasks (V_N): Travel, A_1, A_2, A_3, B_1, B_2

(P):

Travel → A2 B1 [0.2]

Travel → A1 B2 [0.8]

$B_1 \rightarrow A_1\ A_3$ [1.0]

$B_2 \rightarrow A_2\ A_3$ [1.0]

$A_1 \rightarrow$ BuyTicket [1.0]

$A_2 \rightarrow$ GetIn [1.0]

$A_3 \rightarrow$ GetOut [1.0]

Figure 9. Travel domain HTN and SCFG [27].

2.3.5 Computer Networking Traffic Protocol Analysis

Protocol reverse engineering and anomaly-based network intrusion detection are two networking applications that leverage SCFGs. While these problem domains also use computer networking traffic, their terminal set is derived from the networking protocols, rather than the user applications. Therefore, the focus of these efforts is distinct from those proposed in this research.

In protocol reverse-engineering, DeYoung [60] found grammatical inference was possible with Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP3). Similarly, Antunes, et al. [61] conducted protocol reverse engineering on the File Transfer Protocol (FTP). These works indicate that the grammar inference approach is feasible for reconstructing the computer networking communication protocols.

Using protocol definitions, Estevez-Tapiador, et al. [62] developed Finite State Automata (FSA) models of Hypertext Transfer Protocol (HTTP), FTP, and Secure Shell (SSH) and learned the states, transitions, and transition probabilities from captured traffic files using a packet header combinations. FSA models are less expressive than SCFG, but can be represented as SCFG. The authors found that the transition probabilities could indicate network attacks. Sequences with network attacks contained subsequences of low probability transitions. Essentially, they were using a variation of parsing to determine if an attack occurred on a recorded activity sequence, where a parse involving a low probability rule indicates a possibility that an attack occurred.

2.4 Domain Characteristics Suitable for SCFG Representation

Understanding the domain characteristics of a new domain is the first step to applying SCFG to a new domain. A good mapping of domain to representation can provide a means for efficient reasoning and accessibility. A poor mapping can have the opposite effect [37].

Domains must have discrete observables that translate into irreducible symbols in V_T to produce a finite V_T. The V_T size can range from very large, such as in NLP, to very small, such as in bioinformatics. Each V_T symbol must have a corresponding production rule; therefore, V_T cannot be infinitely large. Continuous domains are not suitable for SCFG representation without discretization.

The domain should have an element of causality between observables represented by a V_T symbol. Production rules enforce a linear order on the symbols. Domains that are unordered collections do not benefit from SCFG inference methods that leverage an SCFG's hierarchal structure. Domains where observations depend or anticipate the future also violate the causality assumption, where observances depend only on past observances [39].

The stochastic parameter and the context-free substitution allow SCFG to represent probabilistic domains of infinite size [63]. Therefore, domains that are deterministic with solutions in finite space do not require SCFG; these domains may be represent able using regular expressions or equivalently, finite state automata, which have lower computational complexity than SCFG's inference methods.

While not domain traits, domain data availability and quality also factor into SCFG suitability. An SCFG produces an identifiable representation of the data when only positive examples are available [64]. Positive examples mean that all data samples are accepted by the knowledge representation. In contrast, other representations, such as context-free grammars without stochastic parameters, require negative examples to identifiably represent the domain. SCFG representation is robust against irregular noise patterns in the data, by assigning noisy input with low probabilities [19, 39]. However, if the noise occurs as a frequent and constant pattern within the data, then the SCFG will incorrectly include the noise in the domain representation.

2.5 General Methodology to Apply SCFG on a Novel Domain

The domains that leverage SCFG presented throughout this section serves as starting points for application of SCFG to a new domain, by first identifying a domain that shares similar characteristics. Table 1 lists the domain with their respective characteristics. Three characteristics are marked with an "*"; these characteristics involve SCFG adaptations to the domains, which is discussed at the end of this appendix.

Table 1. SCFG-applied Domains and their Characteristics.

Domain [References]	**Domain Characteristics**
Natural Language Processing (NLP) [13–15]	Defined and discrete V_T ; large V_T to V_N ratio; consistent V_N meaning; linear order (known cross-serialization* in two languages)
Bioinformatics [20–24]	Very small, defined and discrete V_T; V_N variable meanings; linear order; cross-serialization* possible
Activity Recognition [16–19]	Discretized V_T from continuous data; V_N variably assigned or learned; linear order (time); non-linear* order (concurrency) possible
Automated Planning [13, 25–27]	Discrete V_T; V_N variably assigned or learned; both linear order (time); non-linear* (cross-serialization) possible; loops* have inference significance
Computer Networking Traffic Protocol Analysis [60–62]	Discrete V_T; V_N variably assigned or learned; linear order (time); loops* may not have inference significance

2.5.1 Defining Terminal Symbols.

The lowest level data of interest make up the terminal symbols. Continuous data requires a discretization step. This occurs frequently in activity recognition, where the data is analog and an initial step recognizes certain low-level actions and then leverages SCFG to recognize complicated multi-step actions via context. Even with discrete data, clustering may be applied to group low-level data that does not exhibit a causal relationship or to raise the level of detail to a higher level of interest to the domain. NLP part-of-speech tagging is a good example. In applying NLP to plagiarism detection, the part-of-speech groups thousands of unique words. This grouping is then used to recognize part-of-speech patterns instead of attempting to recognize all of the potential word substitutions themselves.

2.5.2 Defining SCFG Production Rules.

Production rules are an important aspect of how SCFG provides domain knowledge interference. This subsection discusses two methods to define the production rule structure, P, and correspondingly the definition of each V_N.

The two approaches to defining grammar production rules include: 1) domain expert definition and 2) machine learning on domain data. For the first approach, a domain expert manually defines each production rule and probabilistic parameters for each rule. This approach is advantageous when a domain expert is available and the domain knowledge is well understood and consistent between data samples. In applications where the data is very noisy, an expert defined grammar can focus the application and grammar to detect only specific patterns of interest. Manual definition is not possible when a domain expert is not available or costly. In addition, this approach is less practical in large problem domains where it is difficult to anticipate and account for the entire spectrum of possible events and outcomes [18]. Furthermore, an expert-defined grammar may be subject to bias, which is problematic if the grammar is intended for an evaluation application [65].

Machine learning on domain data is an alternative to expert-defined SCFG structure. Instead of specifying the rules manually, the domain data is used to create production rules; the goal of the machine learner is to produce a grammar that can parse all entries in the data. Li, et al. [27] presents an algorithm that iterates through the data sample and produces two-right-hand-side production rules. The advantage to using machine learning is that it reduces the reliance on the availability of a domain expert. The machine learning method is heavily dependent on the representative quality of the data set. The two factors affecting data set quality are balance and sampling. *Balance* is the range or scope covered in the domain knowledge and *sampling* reflects the proportion of coverage of aspects of the domain knowledge present in the data samples [14]. Data sample selection is not trivial and the machine learning approach is not without challenges. This production rule learning method also does not handle noise until the production rule likelihood parameter learning stage, where infrequently used production rules with lower likelihood are removed from the grammar, leveraging the assumption that noise is infrequent and random [19].

2.5.3 Adaptations.

Table 2 shows different domains and notes selected references as example application of SCFG to a domain. In some domains, there are multiple applications as indicated in the purpose column, highlighting the versatility of SCFG knowledge representation. To facilitate understanding of SCFG notation, the last four columns translates each of the symbols in the SCFG 4-tuple in domain terms.

Certain domain characteristics complicate SCFG representation. This subsection discusses loops, cross-serialization and nonlinear order and their respective adaptations from past works so that application in novel domains that exhibit similar properties may use or expand on these solutions. Adaptations to hand domain characteristics can occur in probability parameter estimation, parsing methods, or production rule structure. This section is not intended as an exhaustive list of solutions, but rather a starting point and to highlight that domain adaptations may originate from different domains that encounter similar issues.

Table 2. Problem Domains and SCFG Representation.

Domain [References]	**Purpose**	**Terminal Symbol (V_T)**	**Non-Terminal Symbol (V_N)**	**Production Rules (P)**	**Starting non-terminal (S)**
Natural Language Processing (NLP) [13–15]	determine semantics; disambiguate word definitions	words	parts of speech	acceptable language sequence	valid sentence structures
Bio-informatics [20–24]	discover new and/or viable proteins; identify families of proteins	nucleotid es	protein sub-structures	substructure patterns	valid RNA sequence
Activity Recognition [16–19]	identify context of discrete behaviors (larger more complex behavior)	discrete events	sub-activities in linear order	sub activities	most complex activity sequence
Automated Planning [13, 25–27]	(top-down parsing) identify possible actions to meet plan goals (bottom-up) determine if actions fulfill plan goals	discrete actions	sub-activities	activities in linear order	valid plans and planning goals
Computer Networking Traffic Analysis [60–62]	protocol reverse engineering; anomaly detection	packet flags or keywords	partial commands	communicati on protocol sequences	valid protocol usage

Loops.

Loops are symbols or group of symbols that repeat throughout the domain. The presence of loops can have a substantial impact on the inference from an SCFG in two ways. Loops can cause SCFG parameter learning to drop production rules during likelihood estimation because the non-loop sections are sampled less frequently in the data. Dropping the production rules, however, means that the resulting SCFG fails to reflect legitimate domain knowledge. For instance, the computer networking traffic analysis domain that uses packet headers as terminal symbols provides a domain that exhibits loops. In this domain, a data transfer is reflected as loops of `ACK` packet headers. At the end of a data transfer, a legitimate change to connection teardown occurs. `ACKs` may repeat very often, particularly in large data transfers, but in all connections, the teardown sequence only appears once. Machine learning can unintentionally drop the production rules that reflect the connection teardown process because of the low sampling in relative frequency compared to the data transfer loop. Figure 10 below is a packet header transition probability diagram to highlight this domain's looping structure.

Preventing inadvertent production rule pruning therefore requires an adaptation in the parameter estimation phase, involving manual oversight of the machine learning process to ensure domain knowledge does not get lost [27].

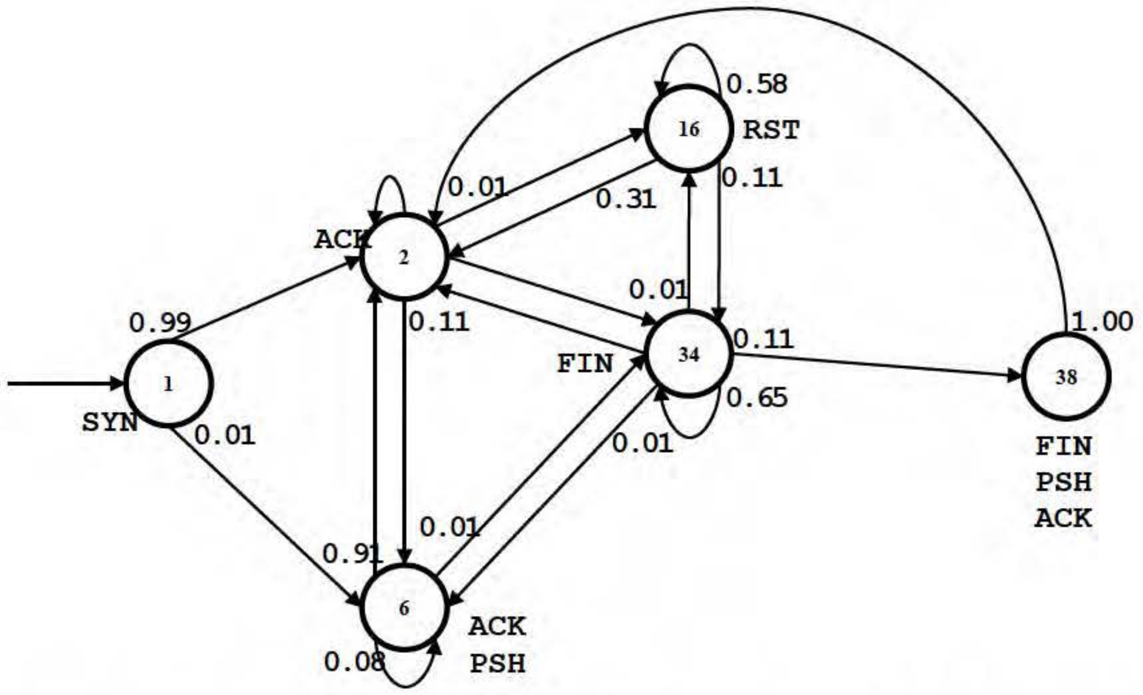


Figure 10. Transition probabilities for Transport Control Protocol (TCP) in the computer network traffic analysis domain [62]. The transitions between the `ACK` and `ACK PSH` states indicate a data transfer, which occurs much more frequently than `ACK` to `FIN`, which signifies connection teardown.

Loops also impact semantic inference from sentence likelihood. Loops, by definition, lengthen sentence length. Sentence likelihood calculated from a product of production rule likelihoods decreases the likelihood with every downward substitution. This effect may be desirable in domains such as planning, where each action takes effort or time, regardless of repetition. However, in domains such as computer networking traffic analysis for anomaly detection, where low sentence likelihood is an indicator unusual network traffic in the data, a drop in likelihood due to data transfer loops does not necessarily reflect unusual event in the domain. In domains where looping events should not decrease sentence likelihood, the adaptation occurs in SCFG parsing methods where using an alternative sentence likelihood method, such as sentence likelihood normalized to length [17], minimum likelihood, or minimum likelihood over a span of *n* symbols [62], may obtain the desired effect.

Cross-serialization.

Cross-serialization occurs when a linkage or dependency exists between symbols that spans over other symbols and no ordering of symbols can remove the span to put the linked symbols together without spanning another symbol. Figure 11 illustrates cross-serialization.

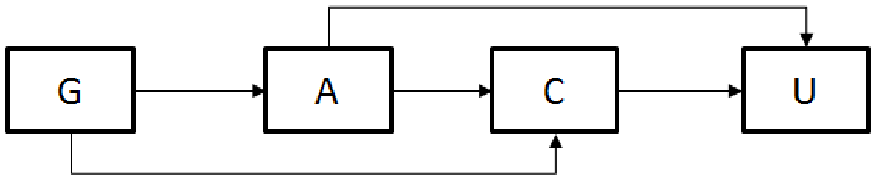


Figure 11. Cross-serialization.

Cross-serialization is not natively expressible in a SCFG due to the constraint that production rules have only a single non-terminal symbol on the left-hand side. However, authors in the planning domain and the bioinformatics domain devised adaptations on the production rule structure and parsing method to express cross-serialization. Geib and Steedman [13] identified instances in plans where cross-serialization exists and propose the Combinatory Categorical Grammar (CCG) that extends SCFG with combinatory rules to provide additional guidance on production rule substitution, without breaking polynomial parsing complexity. In bioinformatics, Rivas and Eddy [20] extended SCFG using a specialized set of non-terminal symbols and a marker symbol (*I*) to tell the parser to switch to cross-serialization handling. A specialized set of rules (*R*) govern the substitutions of the symbols in *I,* after parsing removes all V_N not in *I*. The additional *I* symbols and *R* rules in the grammar increases the time complexity to $O(L^6)$ and a storage complexity to $O(L^4)$, where *L* is the length of the RNA sequence [20].

Nonlinear Order.

Similar to cross-serialization, production rules enforce a strict linear order, where only one terminal symbol is read at a time and sentence parsing is sequential from left to right. Linearity assumes that each terminal is atomic, occurring one at a time. This property works well for domains such as NLP where words are read one at a time or bioinformatics where proteins do not overlap. In domains such as activity recognition or automated planning however, certain actions occur simultaneously or have variable durations.

To increase SCFG expressivity to understand nonlinear order, authors in the activity recognition and automated planning domains introduce logical predicates to relate terminal symbols in a production rule with structure adaptation. Nevatia, et al. [18] defined an ontology that incorporates composite events. Composite events use operators to associate primitive events to recognize multiple agent or non-sequential single agent behaviors. The operators use Allen's interval temporal logic predicates (before, meets, overlaps, starts, during, finishes, equals) to handle relationships that are more than just linearly causal. Ryoo, et al. [55] extended Nevatia, et al.'s three-tier primitive, single-thread, multi-thread hierarchy and included logical predicates to bind other relationships, allowing definition of even more complex, higher level activities. With these adaptations, events in the SCFG are described more expressively, though levels of composite actions leverage production rule substitutions from the grammar hierarchy in the same fashion as simpler non-composite actions.

2.6 Summary

This chapter provided the background to understand the fundamental SCFG concepts. The network forensic applications presented in the next chapter uses grammars for parsing, using the stochastic parameters to make associations. Structure inference is also used in a reduction approach to find the evidence of probative value.

III. Methodology

The previous chapter provided the background on Stochastic Context-Free Grammars (SCFGs) to facilitate discussion on how the application of SCFG knowledge structure can answer the criminal investigative questions through forensic processes. Inman and Rudin's [10] framework comprises of four processes: identification, individualization, association, and reconstruction. This methodology addresses SCFG for association and individualization. Identification is not performed because the problem is scoped to network data and further inference involves subsequent processes. The sequential nature of the PCAP files also provides the information for the reconstruction process than usually available in physical evidence settings.

As discussed in Chapter 2, SCFG parsing and inference methods require discrete, sequential data. After presenting related digital forensic work and general forensic approaches, the methodology starts with a process to turn networking information into activity sequences, or timelines. Following this is the methodology to use SCFG parsing to provide a quantitative measure for association. Finally, SCFG structure inference algorithms identify behavior patterns to sift away explainable events from activity sequences, so the examination can focus attention on activities that are not attributed to normal behavior. The results from association and individualization processes reduce the amount of information in a forensic examination.

3.1 Related Work

Timelines are common in digital forensic procedures. Unlike physical evidence, digital data often comes with time stamps and other meta data that enable timeline construction. This section presents prior work that similarly focus on digital forensic timelines to make an examiner's time and effort more efficient.

Buchholz and Falk [2] designed a graphical timeline editor, Zeitline, to enable manual timeline creation. Their editor allowed the examiners to import evidence from multiple sources and organize them by timestamps. Similar to the concept of SCFG terminals, their design focused on atomic events, which when grouped together, create an event hierarchy with very detailed events such as individual file access at the lowest-levels, to a user task such, as system installation. The tool focuses on time synchronization because inferences of higher-level complex events are based on temporally local events.

Olsson and Boldt [5] similarly identified time to be the most common feature amongst digital artifacts. Timestamps are common and verifiable against other event logs for integrity. To improve upon Zeitline [2], they designed a scanner to minimize the burden of manual data entry. For visualization, their system uses multilevel views that allows the user to view the times where evidence is found at a high level, and then zoom in onto specific occasions, rather than flatly examining every file or event log. In their improvements and future work sections, the authors suggest including data mining and machine learning as methods to help examiners more efficiently identify the interesting parts of the timeline.

Esposito [3] identifies a significant disconnect between timestamps, which are a singular fixed point in time, and timelines which must provide context of activities that occur before, during, and after the event. His examination of the Log2timeline tool is also for the purpose of "cut[ing] through mountains of data to find the needle in the haystack" and his methodology used a selection-based approach, identifying common queries using the Log2timeline tool.

3.2 General Digital Forensics Process

There are two general processes to locate and extract evidence of probative value: selection and reduction [8] . Selection involves targeted and specific queries. With some knowledge of the case, an examiner can use his experience and understanding of the system to deterministically locate what he is looking for. The risk with this approach is that critical evidence may be overlooked and if the examiner has an erroneous hypothesis, he could return with incomplete results [9]. In contrast, reduction attempts to identify collected data that is not relevant. Reduction has the advantage of not requiring case-based searches, but failure to eliminate enough data means wasted time and effort spent on false leads [8]. In the worst case, it may even involve the arrest of the wrong person [9].

The proposed SCFG process uses or identifies general behavioral patterns. These patterns are used in a reductive manner, eliminating suspect pools or routine activities from further evaluation. The process is illustrated in Figure 12, where the processes are denoted as rectangular blocks in the flowchart.

The first process, PCAP to SCFG Terminals, translates the information from computer network traffic into activity sequences. The activity sequences are linear event timelines that are inputs into the two digital forensic applications. With activity sequences from the computer network traffic, the association process uses SCFG to parse the activity sequence with each of the known profiles and outputs the profile that returned the greatest parse likelihood, attributing that activity sequence to the profile, eliminating the other profiles from examination. Individualization also accepts activity sequences as inputs and also leverages a reduction approach. This process uses SCFG structure inference to discover behavior patterns and remove routine behaviors from activity sequences. Individualization outputs the remaining activities in the timelines for further examination.

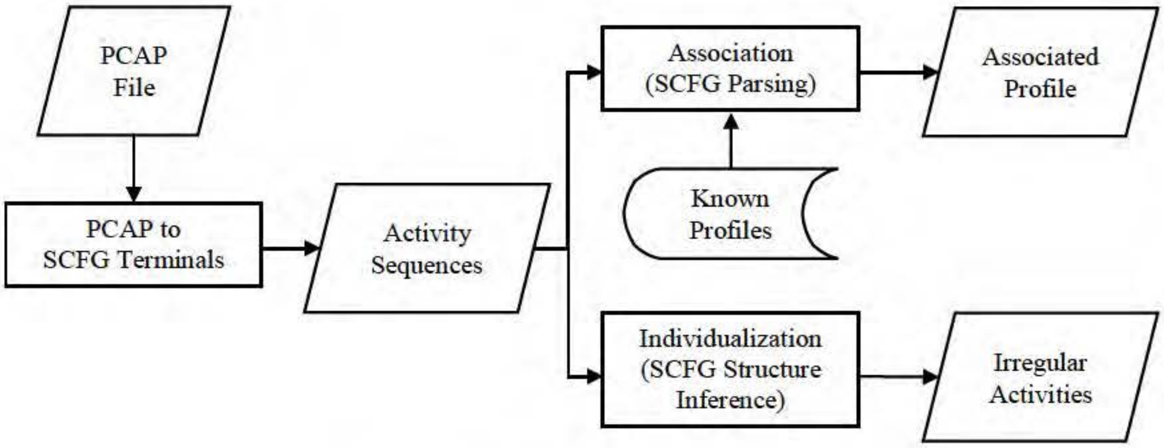


Figure 12. Methodology Process Overview.

3.3 PCAP file

Computer network traffic data in packet capture (PCAP) format exhibits characteristics suitable for SCFG representation. PCAP files records frames serially in time. Even though packets may arrive out of order, programs like Wireshark [66] reconstructs the intended flow and provides a serial order to the TCP streams, even if the packets from the streams interweave.

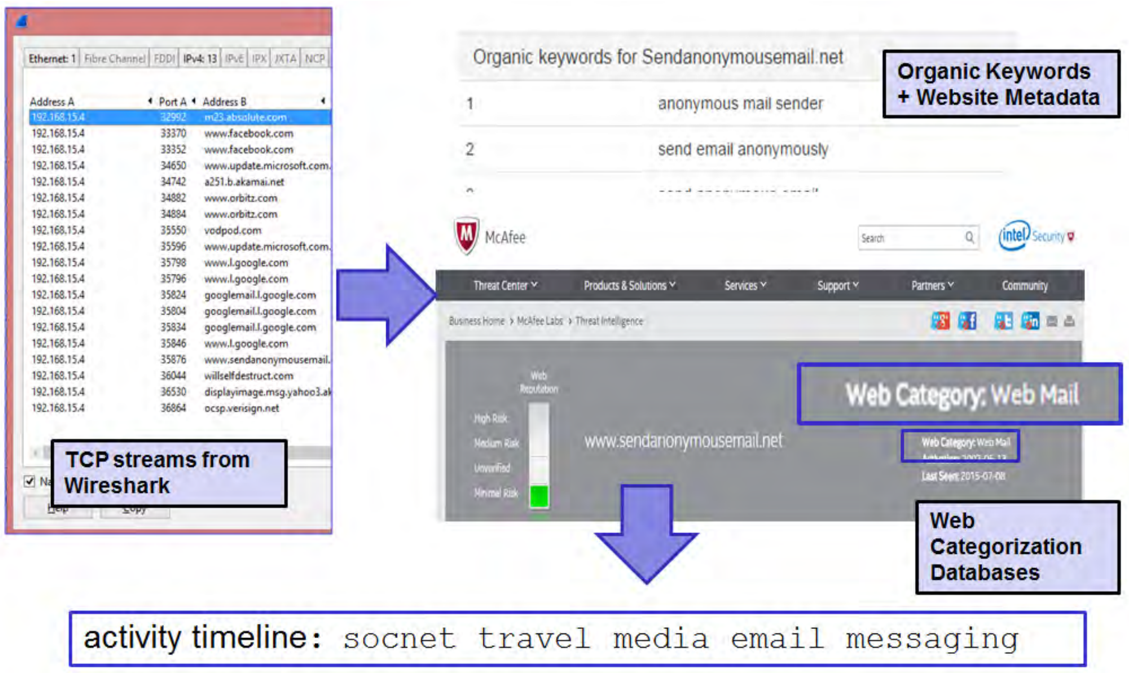
SCFG terminal identification from the flows can be done through port inspection, which can achieve up to 90% classification accuracy [67, 68], organic keywords and website meta data that reveal the category of the websites, or commercial databases. These methods enable classification of a large set of low-level activities to a smaller set of categories, similar to how Natural Language Processing (NLP) reduces every word in a language to a few parts of speech categories.

Focusing on web usage patterns avoids the issues of cookies [69] which requires consent and cooperation from the user. Related work on this includes Yang [70] who attempts user identification using frequent mining measures of support and lift to discriminate between user profiles of web sessions, where the measures indicate the proportion that a pattern appears. Attempting the same with DNS queries, Banse, et al. [71] found user behavior to be stable, though some users did not have enough data for a characteristic pattern to emerge.

Like Mao, et al. [72], we assume that the user does not interfere with the observation process and does not deliberately attempt to defeat the recognition system. Banse, et al. [71] identify additional measures that complicate behavior-based tracking. Anonymizers like Tor obfuscate the destination IP address, therefore preventing classification of destination IP site. However, using anonymizers may be itself a suspicious behavior so detecting Tor traffic satisfies the goal of identifying suspicious activities for a criminal investigator.

3.4 PCAP to SCFG Terminals

Converting PCAP to terminals is the first step to SCFG representation of computer network traffic. Similar to Olsson and Boldt [5], we illustrate the PCAP to terminal process using a digital forensic crime scenario, the fictitious Nitroba University Harassment case [11]. Figure 13 illustrates the conversion of a network capture and the rest of this subsection explains each step in further detail.



Address A	Port A	Address B
192.168.15.4	32992	m23.absolute.com
192.168.15.4	33370	www.facebook.com
192.168.15.4	33352	www.facebook.com
192.168.15.4	34650	www.update.microsoft.com.
192.168.15.4	34742	a251.b.akamai.net
192.168.15.4	34882	www.orbitz.com
192.168.15.4	34884	www.orbitz.com
192.168.15.4	35550	vodpod.com
192.168.15.4	35596	www.update.microsoft.com.
192.168.15.4	35798	www.l.google.com
192.168.15.4	35796	www.l.google.com
192.168.15.4	35824	googlemail.l.google.com
192.168.15.4	35804	googlemail.l.google.com
192.168.15.4	35834	googlemail.l.google.com
192.168.15.4	35846	www.l.google.com
192.168.15.4	35876	www.sendanonymousemail.
192.168.15.4	36044	willselfdestruct.com
192.168.15.4	36530	displayimage.msg.yahoo3.ak
192.168.15.4	36864	ocsp.verisign.net

Organic keywords for Sendanonymousemail.net	
1	anonymous mail sender
2	send email anonymously

activity timeline: `socnet travel media email messaging`

Figure 13. PCAP to SCFG terminal process.

In this scenario, a student sends harassing e-mails to a professor. Examiners seized the network capture, where 11 of her students share the network. This scenario plays out over a 55MB PCAP file containing over 95,000 packets. Only 192.168.1.64 and 192.168.15.4 show significant activity so the activities from these two sources make up the timelines of interest. Applying the "http.request.method==POST" filter trims the timelines to activities that involved user input. Figure 13 shows the TCP stream information from 192.168.15.4. with the applied filter. This TCP stream is the first part of PCAP to terminal process by putting user activities in linear order.

Wireshark's DNS resolver provides human-readable uniform resource locator (URL) addresses for the conversation endpoints. Well-known sites like www.facebook.com are easily recognized as social media. For uncommon sites, examiners can use website meta information and organic keywords. Figure 13 shows the organic keywords for www.sendanonymousemail.net returned from IPaddress.com in the top-right and shows www.sendanonymousemail.net from the timeline through TrustedSource[4]. Proprietary databases, like BlueCoat[5] or TrustedSource, provide large scale URL categorization. The categories are then used as SCFG low-level terminals. Sites attributed to referred ads and background services do not reflect user input so those sites are not included in activity sequences. The TCP streams from Figure 13 resulted in an activity sequence of: `socialnetwork travel media email messaging`

3.5 Association (SCFG Parsing)

Association applies comparisons of competing hypotheses of generalized behavior patterns for the purpose of attributing a profile to the timelines [10]. Performing association requires the set of knowns to provide competing hypotheses. In practice, association rarely identifies the specific offending element, but focuses the investigation by reducing the suspect pool [73]. Association leverages SCFG parsing and the stochastic likelihood to provide the probability of the evidence against the several alternative

[4] https://trustedsource.org/en/feedback/url?action=checksingle

[5] https://www.bluecoat.com/

explanations to enable the examiner to determine which alternative is most likely. Parse likelihood of the sentences by the grammars provide a quantitative comparative measure, that enables the examiner to provide to the investigator, the profile that matched the timelines with the greatest likelihood.

3.5.1 Known Profiles.

Association attempts to use SCFG parse likelihood to determine the originating grammar. Profiles are necessary for the association process. The quantitative result is from parsing the unknown activity sequence with each profile grammar. The profiles also generate the activity sequences for evaluation so the comparison will have truth values. The overall process incurs an integer multiplier to the computational $O(n^3)$ complexity of parsing.

As presented in Section 2.5.2, there are two ways to create SCFGs: expert definition and machine learning. Profiles are in SCFG representation, so the two production rule definition methods apply. The first method is expert definition, where an expert defines the behavior pattern in the profile. The second method uses machine learning on several activities sequences to discover behavior patterns that are converted into production rules. This work uses both approaches: the grammar comparison and association testing uses expert SCFGs, and the individualization testing leverages machine learning.

3.5.2 Grammar Comparison.

Current methods of comparing grammars at the rule-to-rule level [32, 33] or at the output-to-output level [30, 31] are computationally impractical or undecidable. Grammar-based rule-to-rule methods that attempt to replicate and substitute rules to generate equivalent rules are decidable, but exceed polynomial complexity [32]. Output-to-output methods that compare similarity of two grammars using only their outputs are not decidable [33–35]. Appendix A provides related works on grammar comparison. Both techniques have merit in terms of comparing structure or resulting sequences. The novel grammar comparison method combines both of these to perform the comparison with $O(n^3)$ complexity.

The grammar comparison method, shown in Figure 14, applies graph node matching to examine grammar symbol causalities, to identify similarity between grammars. SCFG rules enforce the symbol causalities and the approach uses the causalities as the measure of similarity.

By translating the grammar into a graph, we leverage the advantages of graph-based representation for structure comparison. The graph captures the connectivity relationship of the SCFG production rules into a single summarized presentation. The approach leverages only the causality relationships defined in the production rules and does not incorporate the stochastic parameters. Without requiring the stochastic parameters, the rest of the discussion on grammar similarity treats the SCFG as Context-Free Grammars (CFG). The conversion to graphs produces source and terminal matrics used for the Zager-Verghese graph node-matching algorithm [36]. The graph-node matching produces a node-likeness matrix, containing node likeness scores between nodes across the two graphs. Using the Hungarian algorithm produces a node pairing that maximizes the scores. With the pairings, the grammars now have a way to relate symbols, which is then used to identify common causal patterns between the two grammars as a measure of grammar similarity. This section explains the process and an example grammar comparison is in Appendix A.

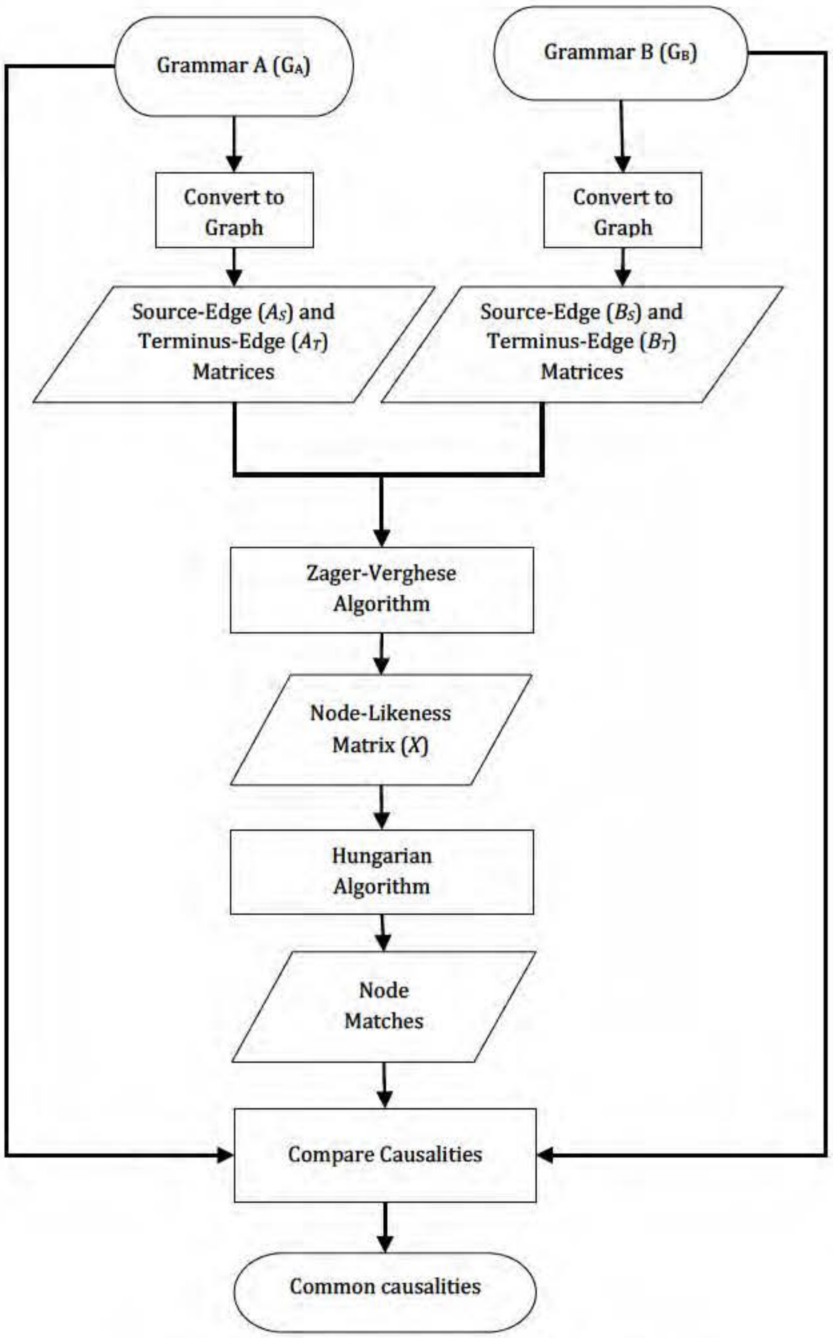


Figure 14. Grammar Comparison Process.

Convert CFG to Graph.

Translating CFGs to graphs is not unprecedented. Muggleton and Pahlavi [74] relate CFG to a stochastic automata, by translating the production rules into states and transitions. Gecse and Kovacs [75] provide another example of translating CFG into graphs, for the purpose of identifying grammar consistency, highlighting a pragmatic benefit of examining CFG in graphical form. For comparison purposes, the proposed method uses a translation similar to Gecse and Kovacs [75], which converts the symbols into states and the links represent a connection between symbols within a CFG production rule. A difference between the approach is that all CFG symbols are represented as nodes in the graph, not just the non-terminals. The graph node matching algorithm used in the proposed approach has an $O(n^3)$ complexity.

Applying a graph node-matching algorithm provides a measure of similarity and compares grammars by matching a symbol in one grammar to its closest approximation in another symbol based on each symbol's connectivity to other symbols. The node-matching enable comparisons regarding the causality of symbols in CFG notation. Similarity is measured as a combination of likeness between symbols and comparison of common causal relationships between symbols, where the existence of a causal link in both grammars indicate similarity while differences in causal links indicate dissimilarity.

The comparison method uses an iterative graph-based approach because it assumes conditions most similar to the grammar comparison problem, where the terminals and non-terminals are not guaranteed to be consistent across grammars. Non-iterative graph-based approaches, such as edit distance/isomorphism or feature extraction, typically assume a shared set of terminals and non-terminals between the grammars under comparison and exceed polynomial complexity.

We first represent each grammar as a directed graph in order to take advantage of graph node-matching. In graph form, each node represents a grammar symbol. The node-matching algorithm then produces a node likeness matrix. Applying the Hungarian algorithm [76] on the node likeness matrix produces a pair-wise matching of terminals between the grammars. Production rules then provide additional causality information that combined with the pair-wise matching, yields insight into grammar similarity. Figure 14 illustrates the grammar comparison process.

Each CFG production rule defines an equivalency relationship between the left hand side (LHS) symbol and the right hand side (RHS) symbols. A graph representation of the CFG also conveys this relationship between LHS and RHS symbols.

The graph nodes correspond to the V_N and V_T symbols. Each production rule creates an edge between the LHS symbol's node and its RHS symbols' nodes. In graph representation, the S node is the node without incoming edges and V_T nodes are nodes without outgoing edges. This is different than the representation in [75] which does not include V_T nodes. Another difference is that edges are numerically labeled instead of their stochastic likelihood. The edge labels are used in a node-edge correspondence to produce a pairwise similarity matrix between nodes. Figure 15 uses the example grammar from Gecse and Kovacs [75] represented as a directed graph. The mathematical operators in the RHS not relevant to the example were removed from the production rules in the figure for clarity.

V_T:	a		
V_N:	S,T,F		
S:	S		
P:			
	S	$\rightarrow$	$S\ T$
	S	$\rightarrow$	T
	T	$\rightarrow$	$T\ F$
	T	$\rightarrow$	F
	F	$\rightarrow$	S
	F	$\rightarrow$	a

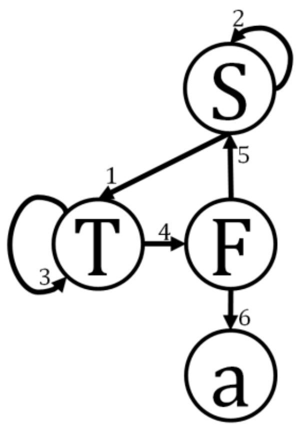


Figure 15. Example CFG and Graph Representation.

The numeric label for each link is used only in the construction of matrices in the source-edge (G_S) and terminus-edge (G_T) matrices, shown in Figure 16, in the Zager-Verghese method [36] for node-matching so their order is unimportant other than consistency between G_S and G_T [36].

G_S	1	2	3	4	5	6
S	1	1				
T			1	1		
F					1	1
A						

G_T	1	2	3	4	5	6
S		1			1	
T	1		1			
F				1		
A						1

Figure 16. Source-edge matrix (G_S) and Terminus-edge matrix (G_T) corresponding to the graph in Figure 15. The non-filled spaces are zero entries.

Zager-Verghese Graph Node-Matching Algorithm.

This approach combines CFG-specific information with the Zager-Verghese [36] iterative graph similarity algorithm. Among the iterative methods, Zager-Verghese is used because it has similar conditions to the CFG comparison problem in that correspondence between nodes is unknown and similarity is calculated on all node pairs between graphs [77].

The Zager-Verghese [36] node-matching algorithm iteratively calculates similarity between nodes applying the assumption that two nodes are similar if their neighborhoods are similar. We selected this graph similarity method because it does not require the grammars to share the same $V_T \cup V_N$ set or labels, in contrast to other graph similarity algorithms [1]. Their contribution to graph node matching is that their algorithm converges independent of initial values. Using G_S and G_T, the algorithm iteratively calculates node-likeness (X) and edge-likeness (Y) scores.

Hungarian Algorithm.

Applying Hungarian algorithm [76] on X produces a lower-bound node matching between nodes across the two graphs. The iterative calculations for X and Y require matrix multiplications that are $O(n^3)$ [78, Ch. 13] and the Hungarian algorithm is also $O(n^3)$ [79].

Causality Comparison.

The node mapping enables comparisons between grammars with different symbols because the symbols are matched based on their connectivity to other symbols. In addition, a CFG in graph form reflects the following CFG-specific information:

1. V_T are the only sink nodes
2. V_N always have at least one out-going link
3. RHS symbols have a causality relationship with one another
4. S only has outgoing links
5. Connectivity to self (1's in the same coordinate in both G_S and G_T) indicate recursion

The graphical representation over-generalizes the CFG and does not take into account all the knowledge represented in a CFG. For instance, the production rule $S \rightarrow A\ B$ and $S \rightarrow B\ A$ are indistinguishable in graph form, which does not denote the causality relationship between A and B. Thus, it is possible to translate a grammar into a graph, but not possible to definitively reconstruct a grammar from the graph without providing the additional causality information.

CFG-specific information can be further incorporated by manually altering X. For instance, if we know that two grammars share a common V_T, we zero out matching scores between different terminals in X to prevent irrelevant node matches. The same approach applies to zeroing out matches between V_N versus V_T if we know certain symbols are definitively in V_N or V_T. All adjustments to X based on additional CFG information occur prior to running the Hungarian algorithm.

The node mapping is used to examine the causal relationships between each grammar. Each grammar has a list of node pairs that specify a before-and-after relationship in the production rule's RHS. After the graph node matching, CFG comparison requires examining the causal commonality between RHS symbols not captured in the summary graph. In doing so, CFG similarity extends Zager and Verghese [36] pairwise node similarity scores to include node causality similarity, rather than a single similarity index. This algorithm is shown in Algorithm 1.

Algorithm 1. CFG Similarity Algorithm.

```
Input:
G_A (Grammar A)
G_B (Grammar B)                    //Grammar B nodes ≥ Grammar A nodes
A_S (Graph A Source-Edge Matrix)
A_T (Graph A Terminus-Edge Matrix)
B_S (Graph B Source-Edge Matrix)
B_T (Graph B Terminus-Edge Matrix)

Output:
List of matched causalities in G_B to causalities in G_A

X′ = ones matrix of size (number of      //buffer for node similarity
nodes in B_S , number of nodes in A_S)   scores
Y′= ones matrix of size (number of       //buffer for edge similarity
edges in B_S, number of edges in A_S)    scores

for n-iterations                         //number of iterations
    $Y = B_S^T X' A_S + B_T^T X' A_T$    //Y stores the edge similarity
    Y = normalize(Y)                     scores
                                         //T superscript is the matrix
                                         transpose operation
    $X = B_S^T Y' A_S + B_T^T Y' A_T$    //X stores the node similarity
    X = normalize(X)                     scores

    X′ = X                               //copy updated to buffer matrices
    Y′ = Y                               for next iteration
end
 X = addCFGinfo (X)                      //modify X with CFG terminal
                                         information, such as zeroing out
                                         cells between terminal and non-
                                         terminal symbols or keeping known
                                         terminal matches
Hungarian(X)                             //generate pairwise node-to-node
                                         matching
$C_A$ = identifyCausalities($G_A$)       //identify symbol causality in both
$C_B$ = identifyCausalities($G_B$)       grammars and store as lists
$C_B$ = remapSymbols(X, $C_B$)           //remap the symbols in C_B using
                                         results from the node matching
return compareCausalities($C_A$, $C_B$)  //compare causalities for
                                         similarities and differences in
                                         symbol causality
```

By convention, G_A is the smaller grammar, where size is determined as the number of nodes. Each grammar symbol uses a subscript of the grammar when it is not obvious. Capital letters denote non-terminal symbols and lower-case letters denote terminal symbols. The node-likeness matrix (X) contains the values after running Zager-

Verghese [36] through 1,000 iterations. In the example in Appendix A, the bold entries denote the pairwise node matches from the Hungarian algorithm. Causal links between symbol pairs are designated with a ">", where it conveys precedence.

Comparisons are run on MATLAB version 12.1a and used the YiCao implementation [76] of the Hungarian algorithm. In contrast to the Borlin's implementation used in [36], YiCao's implementation does not require padding of X and Y. The implementation performs matching by cost, so X was multiplied by a -1 factor to find minimum cost assignment matching.

The Hungarian algorithm performs a node-to-node comparison that does not account for the possibility where a node in one graph may represent multiple nodes in other. It correctly matched the recursive symbol and the resulting node matching showed greater similarity in the causal links than without having the node matching information.

3.5.1 SCFG Parsing.

Parsing is performed using an Earley parser written in C++ [6]. This particular implementation was selected primarily because it was only a parser and did not contain domain dependent functions or methods. It also did not require conversion of the grammars into Chomsky-Normal-Form.

3.6 Individualization (SCFG Structure Inference)

Individualization attempts to discover behavior patterns from specific sequence of events to focus investigative efforts on events not attributable to the normal activity patterns. The application leverages the individual characteristics that make the object

[6] https://github.com/shaobohou/pearley

unique, where the behavior patterns make up the individual characteristics. Broeders [80] uses the example where a scratch on a bullet is not unique, but a specific arrangement of scratches makes it unique. Similarly, an activity is not unique, but frequent patterns of activities become unique to a user.

Pattern discovery uses two unsupervised SCFG inference techniques, alignment-based inference and bigram-based inference. Alignment-based inference is a top-down approach to find overarching patterns across the timelines. In contrast, the bigram approach is a bottom-up approach identifies patterns as mergers of frequently occurring adjacent events. These patterns are used in reduction to focus the examination on events that do not fit a pattern. The examiner can then provide the criminal investigator the unexplained events in addition to the patterns of behavior, which may also be of probative value.

3.6.1 Alignment-based Inference.

Alignment-based structure learning attempts to discover patterns from the top-down by identifying causal patterns of symbols throughout the data. Alignment considers the possibility that the causal patterns may consist of symbols that are not immediately adjacent to one another by allowing gaps in pattern sequences.

Clustering the corpus improves the resulting alignments when aligning similar sequences. Sequence similarity is based on the arithmetic mean of content distance and edit distance to reflect different similarity characteristics in the timelines [81]. Content similarity takes into account bigram patterns, even if they are not in aligned positions. Content similarity is a combination of precision and recall, which are defined as:

$$Content\ Similarity = 2 \times \frac{Precision \times \sqrt{Recall}}{Precision + \sqrt{Recall}} \tag{3}$$

$$Precision = \frac{|bigram(v_1) \cap bigram(v_2)|}{|bigram(v_1)|} \tag{4}$$

$$Recall = \frac{|bigram(v_1)| \cap |bigram(v_2)|}{|bigram(v_2)|} \tag{5}$$

Where:

$bigram(v)$ is the set of bigrams from sentence v

$|bigram(v)\ \cap bigram(v)|$ is total number of common bigrams

Edit distance takes into account similarity when there is alignment, even though bigram patterns are not preserved. The edit distance algorithm is shown in Algorithm 2.

To produce a result that can be used with content similarity, the edit distance is normalized by the length of the longer sentence, shown in Equation (6). This way, both content and edit distance similarity are on a range between 0 (completely dissimilar) to 1 (completely identical). Table 3 shows example calculations for each of the similarity measures.

$$Edit\ Distance\ Similarity = \frac{max\big(length(v_1), length(v_2)\big) - edit\ distance(v_1, v_2)}{\max\big(length(v_1), length(v_2)\big)} \tag{6}$$

$$Combined\ Similarity = Content\ Similarity + Edit\ Distance\ Similarity \tag{7}$$

Algorithm 2. Edit Distance Algorithm.

```
Input:
string1 [1..m]
string2 [1..n]

Output:
distance between string1 and string2
initialize distance_array(m,n)                           //set 2-dim array
                                                         to all 0's
del_err_cost = 1;                                        //cost can be
ins_err_cost = 1;                                        changed to bias
sub_err_cost  = 1;                                       against specific
                                                         error types
for i = 1 to m
      d[i,0] = i
for j = 1 to n
      d[0,j] = j
for j=1 to n
      for i = 1 to m                                     //letters match
             if string1[i] == string2[j]
               d[i,j] = d[i-1,j-1]
             else
                d[i,j] = min(
                           d[i-1,j] + del_err_cost,      //deletion error
                           d[i,j-1] + ins_err_cost,      //insertion error
                           d[i-1,j-1] + sub_err_cost)    //deletion error
                         )
return d[m,n]                                            //distance
```

Table 3. Examples of Similarity.

v1	v2	Content Similarity	Edit Distance Similarity	Combined Similarity
ABC	CAB	0.59	0.33	0.92
ABCD	DABC	0.73	0.50	1.23
ABC	ADC	0.00	0.66	0.66
AAAA	AAAAB	0.93	0.80	1.73

Clustering the corpus reduces the number top-level *S* productions since each cluster represents at most one *S* production. Each cluster is then recursively aligned using multisequence alignment. If there is an alignment, it is turned into a production rule and the parts of the sequences that do not match the alignment are stored as a subcorpus, tracked by a symbol, *N*, that increments with each new subcorpus. The halting condition is when no alignment is found in a corpus or subcorpus. All sequences are associated to the tracking symbol. However, if a subcorpus produces an alignment, then that alignment gets stored as a production rule and the multialignment is performed on the subcorpuses surrounding the aligned symbols. This approach is outlined in Algorithm 3.

The algorithm uses the Needleman-Wunsh algorithm for pairwise sequence alignment from the bioinformatics domain [82]. Similar to edit distance, the Needleman-Wunsch algorithm uses a scoring system that rewards aligned symbols and penalizes gaps and mismatches. A score matrix and a corresponding traceback matrix records the alignment path that determines aligned positions and insertions of necessary gaps. The algorithm is shown in Algorithm 4 and has an $O(mn)$ time and space complexity, where m and n are the length of the two sequences.

Using the Needleman-Wunsh in a progressive manner builds a multiple sequence alignment from a series of pairwise alignments to avoid simultaneous multiple sequence alignment algorithms which incurs an exponential computational complexity of $O(2^k n^k)$, where k is the number of sequences [83, Ch. 6]. The pairwise progression incrementally adds additional sequences to past alignments and back-propagates gaps into previous alignments when gaps are necessary to align the newest sequence. Order alignment has an impact on the overall alignment because of the introduced gaps. A greedy approach to ordering uses a similarity matrix to identify most similar sequences first. Each cell in the similarity matrix is populated with similarity measures such as edit distance normalized to the longer sequence length. Then, the ordering begins with the two-most similar sequences and adds the remaining most similar sequence until the ordering includes every sequence, creating a guide-tree. Pair-wise alignment then uses the ordering to determine the incremental sequence of alignments.

Algorithm 3. Alignment-based Structure Learner.

```
Input:
W - list of all timelines

Output:
P - production rules
----------------------------------------------------------------------------
N = 0                                   //non-terminal index
clusters = cluster(W)                   //number of clusters drive
                                        the number S productions
                                        //cluster function
                                        described in text
foreach cluster in clusters
    find_alignment(cluster,N)           //recursively called
                                        //each cluster becomes a
                                        corpus

function find_alignment(corpus, N)      //corpus is an input
                                        variable for list of
                                        timelines (cluster on 1st
                                        call) or partial segments
                                        of timelines (subcorpus on
                                        recursive calls)
    alignCol = multiAlignment(corpus)   //no aligned columns base
                                        case
    if alignCol.size = 0
        associate each sequence to N in P
        return
    else                                // Example:
        map incremented N to a subcorpus    // 0 …N+1… * …N+2… * … N+3…
             surrounding aligned columns
        associate N to alignment
        add alignment to P

        foreach subcorpus in corpus     // if * are aligned columns
            find_alignment(subcorpus,N) // the …N… becomes a
                                        subcorpus identified by N
```

Algorithm 4. Needleman-Wunsh Alignment Algorithm.

Input:
```
seq1 [1..m]
seq2 [1..n]
```

Output:
```
alignment1 [1..x]
alignment2 [1..y]
alignment_score

score_matrix[m+1,n+1]                                     //create score and
traceback_matrix[m+1,n+1]                                 traceback 2-dim
                                                          matrices
init_penalty = -10;                                       //scoring system
gap_penalty = -2;
match_reward  = 5;
mismatch_penalty = -3;
assignment_score = 0;

for i = 1 to n                                            //initialize score
      score_matrix[i][0] = i * init_penalty               matrix and
      traceback[i][0] = “up”                              traceback matrix
end
for j = 1 to m
       score_matrix[0][j] = j * init_penalty
      traceback[0][j] = “left”
end

for i = 1 to n
      for j = 1 to m
             int s                                        //temp var
             if (seq1[j-1] == seq2[i-1]  s = match_bonus
             else s = mismatch_penalty

             int diag = score_matrix[i-1][j-1] + s;       //identify scores
             int up = score_matrix[i-1][j] + gap_penalty  to determine path
            int left = score_matrix[i][j-1] + gap_penalty direction in
                                                          traceback matrix
            score_matrix[i][j] = max(diag,up,left)        //score_matrix
                                                          records the max
                                                          score
            traceback[i][j] = max(“diag”,”up”,”left”)     //traceback_matrix
     end                                                  records the
end                                                       direction of the
                                                          max score
i = n
j = m
while ([i][j] != [0][0])
     if (traceback[i][j] == “diag”)                       //diag means
          alignment1.prepend(seq1[j-1])                   symbols are aligned
          alignment2.prepend(seq2[i-1])
           i=i-1
          j=j-1
   else
       if (traceback[i][j] == “left”                     //left means a gap
            alignment1.prepend(seq1[j-1])                in the seq2
            alignment2.prepend(“-”)
             j=j-1
      else
          alignment1.prepend(“-”)                        //up means a gap in
          alignment2.prepend(seq2[i-1])                  seq1
           i=i-1
end
assignment_score = score_matrix[m+1,n+1]
return assignment_score, alignment1, alignment2
```

3.6.2 Bigram-based Inference.

To induce an hierarchal structure in the activity recognition domain and planning domains respectively, Peng, et al. [81] and Li, et al. [48] apply the intuition that frequently adjacent terminals are instances of higher-level events. Li, et al. [48] iteratively combines symbols into bigrams, starting with looping symbols and the most-frequent bigram. The inferred grammar then adds a production rule with the bigram as the RHS. A new symbol for the LHS replaces every instance of the bigram in the corpus until the entire corpus is deduced to the start symbol, `S`. An expectation-maximization algorithm, such as inside-outside [63], prunes the grammar of productions that occur less than a set threshold. Algorithm 5 shows a modified version of the algorithm. Depending on the domain, the sort function orders the corpus in a manner that makes the most sense. In planning, the shorter plans are more desirable so the sort function reorders the plan based on shortest length first to capture the bigrams from best plans first. In the network timeline domain, creating bigrams from timelines that are most similar to the other timelines may be an alternative approach, similar to the way multiple sequence alignment builds a guide tree to determine the ordering for pairwise alignment.

Algorithm 5. Full-Coverage Bigram Structure Learner.

```
Input:
Set of all terminal symbols $V_T$
list of all timelines, $W$
Output:
Production Rules, $P$

sort($W$)                                          //pre-sort
for each symbol, $t$ in $V_T$ do
    Create new symbol in $V_N$ and create          //add terminal
    production in form $V_N \rightarrow t$         productions for CNF
    Add production to $P$
end
rewriteTimeLines(W,P)                              // $W$ is now only in
                                                   $V_N$ symbols
while not empty($W$) do
    while length($w$) > 2 do
        add production $Z \rightarrow X\ Z$ for new        //$X$ is repeated
        loops                                      symbol, right-
                                                   recursive format
        rewriteTimeLines(W,P)                      //checks existing
                                                   rules
        add production $Z \rightarrow X\ Y$ for most-frequent-bigram
        rewriteTimeLines(W,P)
    add production $S \rightarrow w$ to $P$                   //do not add if
                                                   already in $P$
    remove $w$ from $W$

normalizeWeights(P)
```

Instead of performing the pruning at the end, Peng, et al. [81] combine terminals when their joint-occurrence frequency is larger than their expected marginal frequency. They build joint frequency and marginal frequency tables from the timelines and apply a chi-square test on each bigram to determine if the bigram should be represented as a production rule. By using the chi-square test, bigram combinations that fail the significance test are not combined and the algorithm reaches a halting condition faster than the full-coverage algorithm. A variation of this algorithm is shown in Algorithm 6, which does not show the generalization function in Peng, et al.[81]'s algorithm for clarity. In addition, the chi-square function will reject bigrams of symbols that exist only as a bigram because too many variables in the equation are zero. However, if the bigram frequently appears in Joint Frequency Table, then rejecting the bigram produces counter-intuitive response based on the semantics of what the structure learner is attempting to accomplish. Thus, in addition to checking for divide by zero values, the chi-square test checks for situations where this occurs. The implementation retains a history of bigram combinations at each level in V_N^L that makes the hierarchy evident. The highest level W^L defines the S productions in the inferred grammar.

Using the chi-square test improves upon coverage-based algorithms like the one in Algorithm 6, but incurs computational complexity. Coverage-based algorithms, such as SEQUITUR [84] can achieve linear time and space performance--the algorithm efficiently adds new bigrams but does not revisit and reorder combinations, other than to enforce two properties that guide rule usage.

The bigram approach however is sensitive to the rewrite process. For instance, if `AB` and `BC` are both significant bigrams, `ABC` can be written as either `(AB)C` or `A(BC)`. Grouping sensitivity potentially obscures patterns at higher levels. For this reason, bigram inference is complimented with alignment-based inference for this forensic application.

Algorithm 6: Chi-Square Test Bigram Structure Learner.

Input:

```
merge-threshold, m
list of all timelines, W
```

Output:

```
list of combined activities, $V_N^L$ at level L
$W^L$, rewritten with $V_N$ at each level L
```

```
L = 0                                             //terminal level
$V_N^0$ = terminals(W)                            //$V_N^0 = V_T$
do
   L++                                            //R is a map of the
   ($W^L$,$V_N^L$) = collocation($W^{L-1}$,$V_N^{L-1}$,L, R)   bigram to the first and
until $V_N^L$.isEmpty()                           second symbol

function collocation($W^{L-1}$,$V_N^{L-1}$,L, R)
   foreach bigram in $W^{L-1}$
        increment bigram count in jft             //jft stores the
                                                  frequency of the bigram
        increment bigram count in mft1            //mft1 stores the
                                                  frequency of a bigram
                                                  containing the first
                                                  symbol
        increment bigram count in mft2            //mft2 stores the
                                                  frequency of a bigram
                                                  containing the second
                                                  symbol
   end
   T = total number of bigrams in jft
   A = bigram count in jft                        //chi-2 shortcut for 2x2
   E = mft1 count of bigram’s first symbol        checks
   G = mft2 count of bigram’s second symbol
   C = E – A                                      //must check for div by
   B = G – A                                      zero error
   F = T – E
   H = T - G                                      //chi-2 invalid if less
   D = F - B                                      than seven of the
   chi = $T*((A*D)-(B*C))^2$ / (G*H*E*F)          variables are not zero
   if (chi ≥ m)
         add bigram to $V_N^L$                    //for domain purposes, if
         add bigram to R                          symbols in bigram only
   rewrite $W^L$ in $V_N^L$                       exist together, mark it
                                                  significant
return ($W^L$,$V_N^L$)
```

3.7 Summary

This section presented the algorithms developed for applying SCFG to network forensic applications. The first part of the chapter uses domains with SCFG applications to identify domain traits that suggest suitability for SCFG representation. We then show how PCAP files of capture network traffic can be converted into timelines of a terminal alphabet using IP meta data, organic keywords, and URL classification databases on a network forensic scenario. While we were able to generate a timeline, the PCAP did not contain sufficient information to build an SCFG structure. To demonstrate association with SCFG, we designed four grammars to act as competing hypotheses. Before performing association, we compared the grammars at the output level using terminal frequency analysis and at the grammar level, using graph-based node-matching approach, developed in the course of this research. We then examine SCFG for individualization, presenting the structure learning algorithms that focus on repeated patterns. The next chapter presents the analysis and results from the experimental setups proposed in this chapter.

IV. Analysis and Results

Chapter 3 presented the SCFG algorithms for the network forensic association and individualization. This chapter presents the experimental setup, the results, and analysis. SCFG parsing associates an activity sequence to a known profile, reducing unlikely profiles from suspects under consideration. SCFG structure inference discovers normal behavioral patterns from a series of activity sequences, enabling the examiner to focus on events in the activity sequences that are not explained by the discovered patterns.

This chapter begins with the association process, which describes the design of the known profiles, the results of the grammar comparison to confirm differences between the profiles, and the confusion matrices of associating a sequence to a profile from a set of known profiles. Then, the individualization processes is demonstrated using a computer network traffic use case; SCFG reduced the number of activities with probative value across activity sequences using SCFG structure inference techniques, while retaining the event of interest.

4.1 Association

The association process requires an activity sequence and a set of known profiles, characterized by different behavioral patterns, represented as an SCFG. This subsection first discusses the design of the known profiles. Following this is confirmation of similarity and differences between profiles through the output-to-output and grammar-to-grammar comparison. Finally, the confusion matrices from total parse likelihood and most-probable parse likelihood show that these quantitative measures associate the correct, originating profile.

4.1.1 Known Profiles.

In addition to timelines from a PCAP, the association process in Figure 12 shows an input of known behavior profiles. As mentioned in 3.5.1, the methods to create profiles in SCFG are through expert definition and machine learning. The original plan to create known profiles attempted to use machine learning on the computer network traffic captures from various digital forensic scenarios, like the Nitroba scenario [11]. However, the classroom examples often resolved into a single timeline of unstructured events, which was insufficient to infer behavior patterns, such as the example shown in Figure 13.

We attempted the PCAP to SCFG Terminal process using the PCAP 110 file from the 2013 Digital Forensics Network Challenge and obtained similar results, where user interaction was primarily in the form of GET requests. The file consists of 5,666 packets. Wireshark identifies 176 TCP connections, where all connections originated from a single IP address. The parameters of the scenario was focused on deciphering packet level details, inferring that the entire PCAP is one timeline, so the structure learning could not identify significant patterns.

The Network Trap and Trace scenario from the 2011 Digital Forensics Challenge provided more variation. Only 3,365 packets long, Wireshark identified the second and last TCP conversation as an MIRC connection. The context of the scenario is to identify the intent and actions of the subject and subjects. Considering that the only communication connection occurs in two connections, relevant information is probably contained in those two streams. Concerning target identification, the sites visited between the two communication events are suspect.

The DFRSW 2008 challenge consists of two components where the network capture is only one piece of the evidence. This scenario better reflects realistic scenarios where the crime is not fully encapsulated in one file and network examination may provide only part of the narrative. In the scenario, Wireshark DNS resolution made association of a category to banking and webmail sites straightforward, though some of the IPs did not produce results with any of the IP classification methods mentioned earlier.

The digital forensic scenarios show that network traffic captured as PCAP files exhibit the discrete observable, linear order characteristics suitable for SCFG representation. The variability and the potential for volume of activities also warrants SCFG representation. However, the lack of multiple timelines in these scenarios hindered the ability to use SCFG structure learning to identify patterns in the timelines.

As an alternative, four grammars shown in Figure 17 serve as a set of generalized behavior patterns to provide the competing hypotheses necessary for this application. The four grammars share a common seven-terminal alphabet: (`email, social, news, shopping, travel, wiki, scholar`). The purpose of using a terminal alphabet is to focus on behavioral patterns, rather than individual sites. For instance, the Nitroba scenario used a URL that resolved to www.sendanomymousemail.net, which using a selection-based approach, identified the logical starting point for the investigation.

Oracle A represents an ideal profile where behaviors are rigid with discernable patterns. This behavior is driven by the single `S` production. The variability from this profile comes from intermediate non-terminals, which include recursive productions with have multiple terminal derivations. For instance, `Update` can be any combination of `social` and/or `news` observations, but it must occur between some form of `Comm` and `Task.`

Oracle A (idealized patterns)	Oracle B (random)
`1.00 S --> Comm Update Task`	`0.80 S --> S Task`
`1.00 Comm --> email`	`0.20 S --> Task`
`0.20 Update -> social`	`0.20 Task --> email`
`0.20 Update --> news`	`0.20 Task --> wiki`
`0.30 Update --> Update social`	`0.20 Task --> scholar`
`0.30 Update --> Update news`	`0.10 Task --> news`
`0.40 Task --> LitRev`	`0.10 Task --> shopping`
`0.10 Task --> TOrders`	`0.10 Task --> travel`
`0.20 Task --> Comm`	`0.10 Task --> social`
`0.30 Task --> Task Task`	
`1.00 TOrders --> shopping travel`	
`0.20 LitRev --> LitRev wiki`	
`0.20 LitRev --> LitRev scholar`	
`0.60 LitRev --> scholar`	
Oracle C (pattern in noise)	**Oracle D (multiple patterns)**
`1.00 S --> Task1 Task2 Task3`	`0.80 S --> Comm Task`
`0.80 Task1 --> email`	`0.20 S --> Update`
`0.20 Task1 -- Noise email`	`1.00 Comm --> email`
`0.80 Task2 --> social`	`0.20 Update -> social`
`0.20 Task2 --> Noise social`	`0.20 Update --> news`
`0.80 Task3 --> scholar`	`0.30 Update --> Update social`
`0.20 Task3 --> Noise scholar`	`0.30 Update --> Update news`
`0.10 Noise --> Noise Noise`	`0.40 Task --> LitRev`
`0.10 Noise --> social`	`0.10 Task --> TOrders`
`0.10 Noise --> email`	`0.20 Task --> Comm`
`0.10 Noise --> wiki`	`0.30 Task --> Task Task`
`0.10 Noise --> scholar`	`1.00 TOrders --> shopping travel`
`0.10 Noise --> shopping`	`0.20 LitRev --> LitRev wiki`
`0.10 Noise --> travel`	`0.20 LitRev --> LitRev scholar`
`0.10 Noise --> news`	`0.60 LitRev --> scholar`
`0.20 Noise --> space`	

Figure 17. Oracle Grammars.

In contrast, Oracle B is a shallow and thus, non-descriptive profile. There are two `S` productions, but only to describe the loop of `Tasks`, which can be any low-level terminal. Essentially, this oracle generates timelines of random activities, controlled by the looping probability and probability of `Task` to each respective low-level terminal. Through this grammar structure, Oracle B has coverage over any sentence generated with the common, seven-terminal alphabet. The purpose of this oracle is to compare the effect of a general grammar against oracle grammars that exhibit patterns.

Oracle C has a defined sequential pattern of `email`, `social`, and `scholar` observations. Unlike Oracle A, the pattern is intentionally intermixed with other symbols to represent noise using the `Noise` non-terminal, which mimics the * noise non-terminal in Kitani, et al. [19]'s experiment.

The purpose of Oracle D is to have a profile that shares many of the same production rules as Oracle A. Oracle D exhibits two strict pattern, and is the only profile to have multiple patterns with more than one `S` production.

The oracles share a common alphabet to avoid the situation where the presence or the absence of a terminal is sufficient for association. To verify the output characteristics, we graph the frequency of terminals in the corpuses of 100, 1000, and 10,000 activity sequences generated by the oracles. The oracles under comparison use a common terminal alphabet and the purpose of this step is to examine whether terminal frequencies reveal the oracle that created the corpus. Figure 18 plots the frequency of each terminal in the corpus, normalized to the number of timelines in the corpus, as indicated in the parenthesis. Corpuses A, C, and D exhibit a tight frequency bands independent of the number of timelines. As expected from the production rule likelihoods, corpuses from Oracle B do not follow a consistent pattern, as expected given the randomness driven by

Oracle B's production rules. As desired, the frequency lines are intertwined and the presence of a symbol does not clearly identify a specific originating corpus.

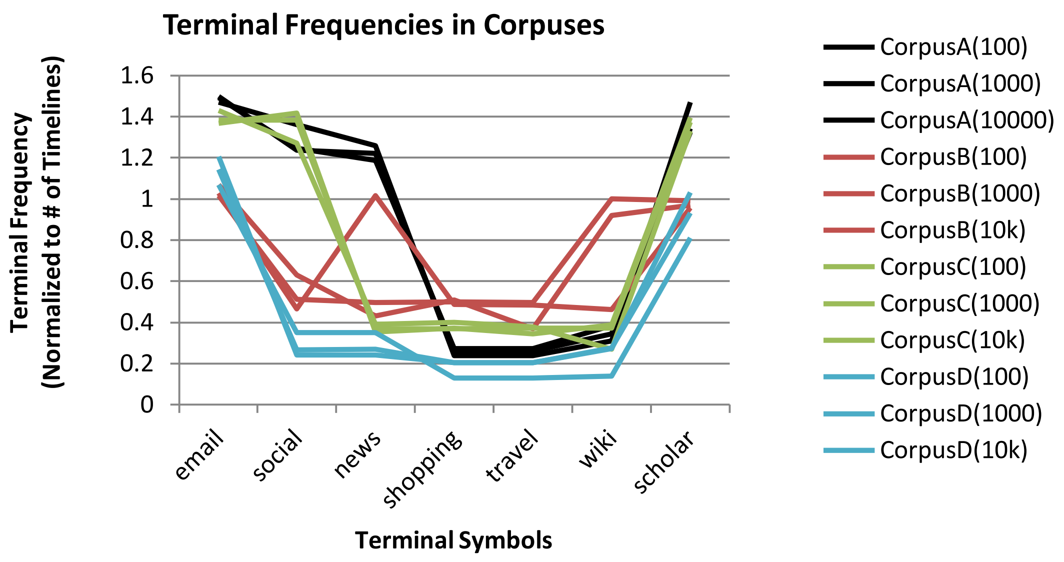


Figure 18. Terminal Frequency Timelines.

4.1.2 Grammar Comparison.

Grammar comparison is performed where it is expected that Oracle A, B, and C are distinct, while Oracle A is similar to Oracle D. Table 4 shows the causalities in the four grammars, where ">" symbolizes precedence. The presented grammars (CFG) method calculates the similarity between CFGs by calculating the similarity between the symbols, where symbol similarity is measured by their inclusion and causality in each production rule.

Table 4. Grammar Causalities.

Oracle A	**Oracle B**	**Oracle C**	**Oracle D**
Comm > Update Update > Task Update > news Update > social Task > Task shopping > travel LitRev > wiki LitRev > Scholar	S > Task	Task1 > Task2 Task2 > Task3 Noise > email Noise > scholar Noise > social Noise > Noise	Comm > Task Update > news Update > social Task > Task shopping > travel LitRev > wiki LitRev > Scholar

In comparing Oracle A to Oracle B, the grammar comparison identified non-terminal correspondence between S_B to S_A and $Task_B$ to $Task_A$ that did not produce common causalities. Oracle A compared to Oracle C produced a mapping of $Comm_A$ to S_C, Update to Task1, S_A to Task2, LitRev to Task3, and $Task_A$ to Noise; this mapping yielded only a single common causality for TaskA $\rightarrow$ $Task_A$ from Noise $\rightarrow$ Noise. Oracle A and Oracle D had a common graphical representation and share six common causalities, achieving the desired effect.

4.1.3 SCFG Parsing.

Parsing provides a quantitative comparison between behavior profiles to identify the most likely origin. A sequence was attributed to a profile based on which profile produced the total parse or most-probable parse likelihood. Table 5 and

Table 6 show the confusion matrices for corpuses generated from the grammars against the set of grammars. Most-probable parse likelihood provided greater separation from mis-association as Oracle B, which was designed for coverage. However, this increased the number of mis-associations of randomness to Oracle C and D.

Table 5. Confusion Matrix based on Total Parse Likelihood.

Total Parse Likelihood	CorpusA	CorpusB	CorpusC	CorpusD
Oracle A	**100**	1	0	0
Oracle B	0	**93**	5	9
Oracle C	0	0	**95**	0
Oracle D	0	6	0	**91**

Table 6. Most Probable Parse Likelihood.

Most Probable Parse Likelihood	CorpusA	CorpusB	CorpusC	CorpusD
Oracle A	**100**	1	0	0
Oracle B	0	**91**	0	0
Oracle C	0	1	**100**	0
Oracle D	0	7	0	**100**

4.1.4 Discussion

Parse likelihood and most-probable parse likelihood correctly associated the originating profile. Parsing produces a quantitative measurement for comparison, but the parse likelihood is useful only in comparison against parse likelihoods by other grammars in the set. The value is not useful in determining the association without values from other grammars for comparison. As shown in the confusion matrices, false positives occur. False associations are problematic because it could result in an erroneous arrest. Most probable parse likelihood made more correct associations. Most-probable parse only must be less than or equal to total parse likelihood, providing additional example that the magnitude of the likelihood score is insufficient to make proper associations.

The following paragraphs discuss similarities of this association to anomaly detection and signature-based intrusion detection, which are also computer networking traffic problems. SCFG parsing incurs polynomial complexity and is better suited for data analysis, rather than real-time systems. In addition, the rate of false positives requires an element of human supervision.

Anomaly Detection.

Despite functional similarities with anomaly detection, the association process assumes a human-in-the-loop due to a high false positive rate. Prior work on inferring user behaviors from network usage achieved 60-80% correct associations of activities with the original user [70, 71], which is inappropriate in a real-time, unmonitored setting.

In the digital forensics setting, the most-probable parse is more advantageous from a human-monitoring standpoint since it requires examination of only a single parse tree per profile. Total parse likelihood, in contrast, requires determination of all possible parses, which is difficult to track manually, to ensure that all combinations are accounted for.

Parsing for Flexible Signature-based Intrusion Detection Analysis.

Oracle C was designed to accept a specific pattern in the presence of noise, enabling a break in causality between key pattern symbols. Skoudis and Liston [85] assert that most attacks follow a general five-phase approach consisting of: 1) Reconnaissance, 2) Scanning, 3) Gaining Access, 4) Maintaining Access, and 5) Covering Tracks. These make an overarching pattern for an attack and each of the phases can be implemented in a variety of ways on a lower level. For a grammar, each phase is essentially a `Task` and a network attack is, at the highest level, a five `Task` causal pattern. Gorodetski and Kotenko's [86] ontology, shown in provides examples of how each phase of an attack can be implemented satisfied to fulfill the `Task`. For example, TCP connect scan, UDP scan, Network Ping Sweeps are lower-level activities that can be substituted with a `Reconnaissance Task` non-terminal. The flexibility in using a grammar-based approach is that the lower-level definitions can be modified or changed without impacting the knowledge structure unrelated to the changes. If network attack grammar can parse the sequence, then the sequence includes all five phases of an attack in the correct order. Using grammar parsing in this manner is different than making comparisons between profiles, but it does meet the description of the association process where parsing identifies the modus operandi.

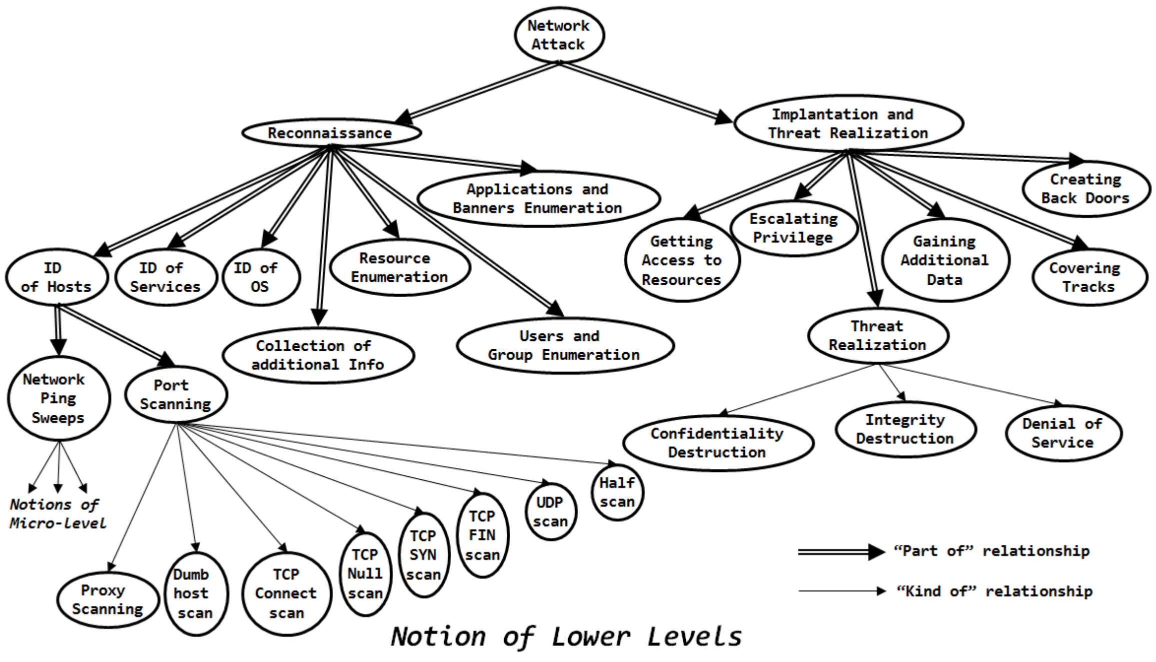


Figure 19. Network Attack Ontology [86].

4.2 Individualization

We captured a single user's traffic data over the course of three days. Truth data was recorded so that the actual event timeline was known. Wireshark's [66] dumpcap utility was used to capture the packets. Due to out of memory errors, dumpcap started to drop packets and eventually crashed during the end of captures. With the recorded truth data, the sessions were performed again. The grammar inference techniques do not use visit duration as a feature in pattern discovery so the resulting activity sequences were unaffected. The captures were recorded in 15 MB, 18 MB, and 23 MB PCAP files.

For testing purposes, the user purposely visited www.HSBC.com during the middle of the second capture. The site was selected because it is not suspicious site based on the URL; the TrustedSource database categorized it as minimal risk with a banking web category. However, the act of visiting this particular page was in contrast to normal habits which typically carries out banking tasks with shopping activities. The site is also not uniquely identifiable based on visit frequency. With the exception of visits to cacwebmail.afit.edu on Internet Explorer, all other browsing was conducted on Chrome. The goal of the reduction, using patterns discovered in the individualization process, is to eliminate patterned activities while not eliminating the HSBC visit.

4.2.1 PCAP to SCFG Terminals.

The first step converts the three PCAP files into timelines. Sequential streams from the same address were grouped together. This is similar to the loop compression SCFG adaptation to focus the behavior patterns on transitions between activities. Similar to the reconstruction process, the behavior patterns require relative rather than absolution time order [10]. This also abstracts away the length of time spent on an activity, which is not a feature represented in SCFGs. The HSBC activity highlighted in red in the second timeline denotes the uncharacteristic activity. The reduction process should eliminate other events without reducing this activity.

For the same reasons discussed in the PCAP to timeline section, the timeline does not include activities from ad services, typically indicated by the referred-from field in the stream. The timelines also do not include activities caused by background services such as antivirus updates or operating system updates because they are not user initiated. Figure 20 shows the activities from the three sessions, t1, t2, and t3, consisting of 41 total activities.

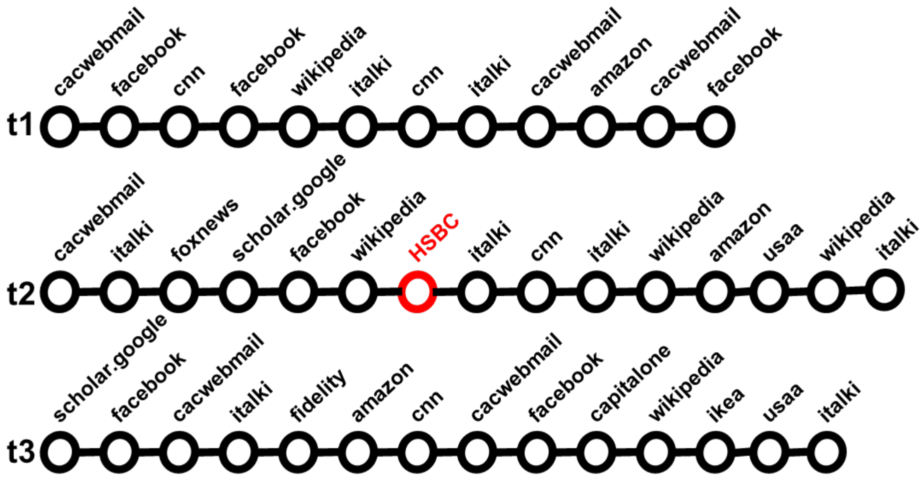


Figure 20. PCAP activities in timeline format of three sessions.

The second step uses IP address meta data and organic keywords retrieved from ipaddress.com as well as McAfee's Threat Intelligence database at www.trustedsource.org to classify the different activities into terminals. Figure 21 shows the timelines as sequences of terminals from a five symbol set of `edu` (education), `socnet` (social networking), `news`, `shopping`, and `banking`.

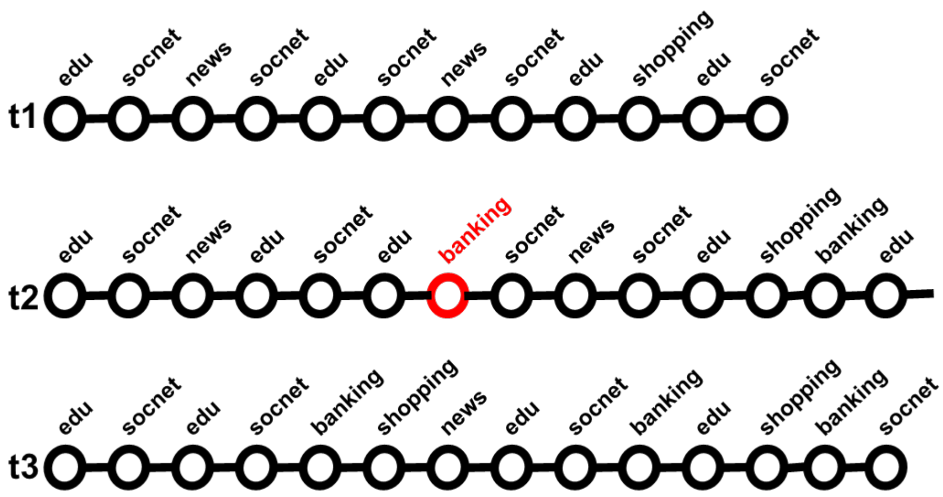


Figure 21. PCAP timelines to SCFG terminals.

4.2.2 *Alignment-based Inference.*

The next step applied alignment-based inference to identify activity patterns that occur across the timelines. This produced an alignment, where aligned symbols are shown in blue in Figure 22. The grayed out dashes represent gaps in the alignment which may include any number of symbols in the timelines.

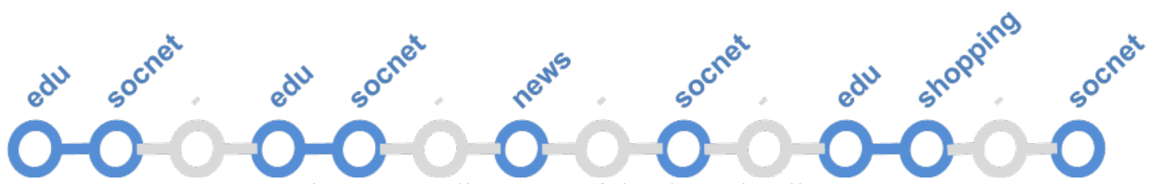


Figure 22. Alignment of the three timelines.

The alignment indicates a pattern of regularity between the timelines. With knowledge of this behavioral pattern, the aligned activities are grayed out to de-prioritize them for investigation, as shown in Figure 23 reducing the total of unexplained activities from 41 to 14.

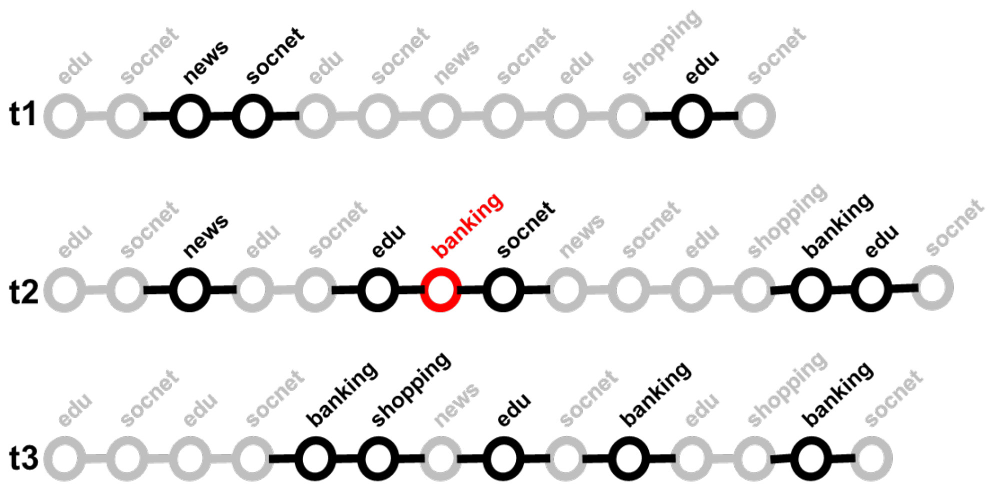


Figure 23. Timelines with aligned symbols de-prioritized (grayed out).

4.2.3 Bigram-based Inference.

Bigram inference techniques further identify frequent activity patterns, focusing on increasing patterns of adjacent symbols. Removing these patterns from the sequences again reduces the number of remaining activities by de-prioritizing activities that frequently occur together. Bigram inference produced the following vocabulary:

- `shoppingbanking,`
- `socnetnews`
- `edusocnet`
- `socnetedu`
- `newssocnet`
- `(edusocnet)news`
- `(shoppingbanking)socnet`
- `edu(shoppingbanking)`
- `(socnetedu)(shoppingbanking)`
- `(socnetedu)shopping`
- `(edusocnet)banking`

The vocabulary terms signify additional behavioral patterns. The parentheses indicate a previously merged bigram within another bigram. Activities that are unexplained by the alignment are matched against the vocabulary list. Sequences that appear in the vocabulary list are also de-prioritized, shown in washed-out green in Figure 24. The bigram discovery process is independent of the alignment inference process. Therefore, the results from the bigram process can reduce the event sequences on their own. By using both approaches, activities can be explained away using both methods. An activity exclude through alignment can still be used as part of a bigram to exclude activities not explained by the alignment discovery process. To highlight these occurrences, activities as part of bigrams that were grayed out in the alignment step are relabeled green, but retain the grey circle.

In comparing results between the two processes, the alignment included two adjacent activities, `edusocnet`, that was also discovered in the bigram inference approach. The alignment also included another adjacent pair, `edu` and `shopping`, which did not appear on the bigram vocabulary list. However, `edushopping` appears in the vocabulary list three times, under `edu(shoppingbanking)`, `(socnetedu)(shoppingbanking)` *and* `(socnetedu)shopping`, marking it as part of other frequent patterns.

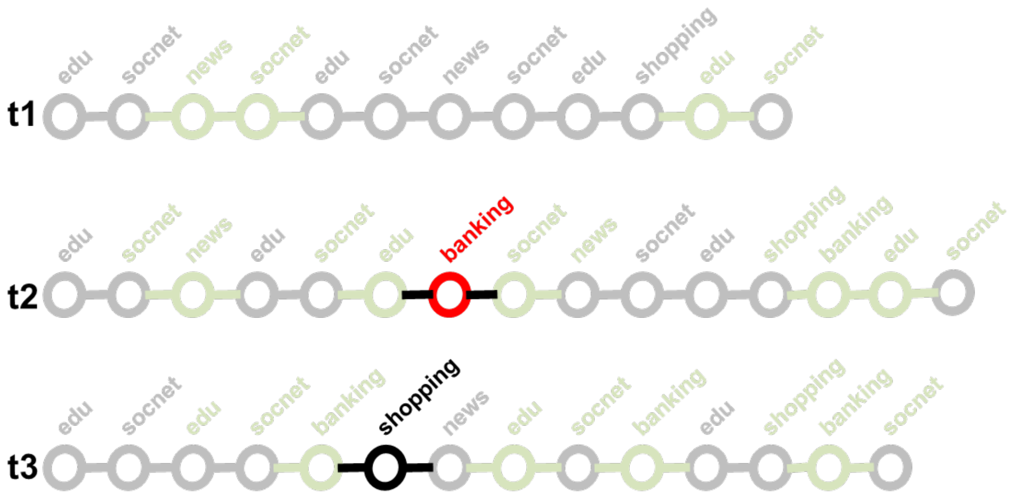


Figure 24. Timelines with both aligned (gray) and bigram (green) activities de-prioritized.

At the point of investigation in Figure 24, only two activities remain across the three timelines that are unexplained by behavior patterns inferred from alignment and bigrams. These two activities should be the start of the investigation, which includes the intended implanted event. This application of SCFG inference primarily leverages the pattern discovery elements of structure learning which was more important than the final inferred SCFG structure to the forensic application.

4.3 Discussion

The application of alignment-based and bigram-based inference reduced the amount of activities requiring more in-depth examination. This section examines variations to the procedure, which highlights future considerations for applying this process.

The first variation deals with the PCAP to SCFG terminal step that eliminates background services from timelines, and the effect on the two structure inference techniques if background services are mis-construed as an user event. Next, the bigrams are evaluated for semantic significance. Finally, the individualization process is examined using six terminals instead of five in the PCAP to SCFG terminal step.

4.3.1 Background Services.

The PCAP to terminal step ignored background services. Inclusion of these sites into the timeline increases the terminal set and the variety of patterns within timelines. Background services that only periodically check for updates is similar to random noise and the terminal will unlikely achieve bigram significance though their occurrence may bisect significant bigrams, requiring more of the occurrences of non-bisected bigrams to achieve the significance threshold. If the background services occur at the same time during user session, such as updates at 7 am on Tuesday, then it may become part of an alignment which gets sifted out as routine. Assuming ad sites are always associated with the referring page, ad terminals would associate with the intended referring page in bigram inference, thereby making them frequent and routine.

In examining the effects of noise and variation, we performed both types of alignment on the corpuses generated in the association section. We examined the bigram inference method, first on 10 and then 100 activity sequences. In Corpuses A, the bigram vocabulary greatly increased with the number of samples, which can attributed to the recursive `Task` symbol that interjects variety at the end of the sequences. Corpus B, experiences a decrease in the number of vocabulary layers and significant symbols. This effect is due to the randomness of Corpus B, where fewer bigrams pass the chi-square threshold because symbols are randomly adjacent to one another. Inferred vocabulary from Corpus C greatly increased with the number of samples, similarly due to the recursive `Noise` production. The graph node matching identified the similarity between

$\texttt{Task}_A$ and $\texttt{Noise}_C$. Corpus D did not experience as large a growth in vocabulary size because the split of ***Update*** meant that the social and news terminals never intermix with other terminals, reducing the number of bigram potential.

Bigram inference is not a good method to identify overarching patterns. The number of overarching patterns is directly correlated to the number of S productions. Table 7 shows the number of S productions using both bigram and alignment-based inference. In contrast, alignment-based inference, which processes the corpus top-down, generated fewer S productions proportional the number of samples. The exception to this is the random patterns in Corpus B, where the number of S productions increased.

Table 7. Number of S productions.

	Bigram-Inference		Alignment-based Inference	
Corpus	# of S productions (10 Samples)	# of S productions (100 Samples)	# of S productions (10 Samples)	# of S productions (100 Samples)
A	9	79	9	48
B	6	84	10	94
C	9	87	10	52
D	7	37	8	33

4.3.2 Bigram Semantics.

The bigram inference identified frequently adjacent activities. However, the vocabulary does not necessarily infer a semantically-significant higher-level activity. For instance, the edusocnetnews pattern indicates that these items appear adjacent to one another but it is not obvious regarding why edu occurs before socnet or socnet before news events. Significance is clearer when reverting the SCFG terminals back to the actual websites. For instance, socnetnews appears because posts and discussions on social networks refer to current events. Therefore, this bigram may carry a semantic meaning of “getting current events.” In contrast, `edusoc` does not convey a higher-level goal. In this sense, the occurrence of these patterns may be an indicator of user preference, rather than some specific plan to accomplish a set task. The inability to guarantee that bigrams have a significant semantic meaning reduces the benefit of using a hierarchal knowledge structure.

4.3.3 Changing an SCFG Terminal.

The `cacwebmail` activity had an afit.edu extension and was assigned the `edu` terminal based on the ipaddress.com classification. We repeat the individualization process and obtain a reduction that categorized `cacwebmail` activity with a `comm` terminal, as a better reflection of the actual activity. This expanded the number of terminals in the sequences to six. As expected, increasing the variety of symbols increases the distance between sequences and this change produced two clusters: (t1, t2) and (t3). An increase in the number of clusters means that there is less likely to be a single alignment pattern, because the alignment pattern from one cluster does not transfer to other clusters. This is illustrated in Figure 22, where t3 does not have any activity in grey; all washed out activities in t3 are due to bigram patterns.

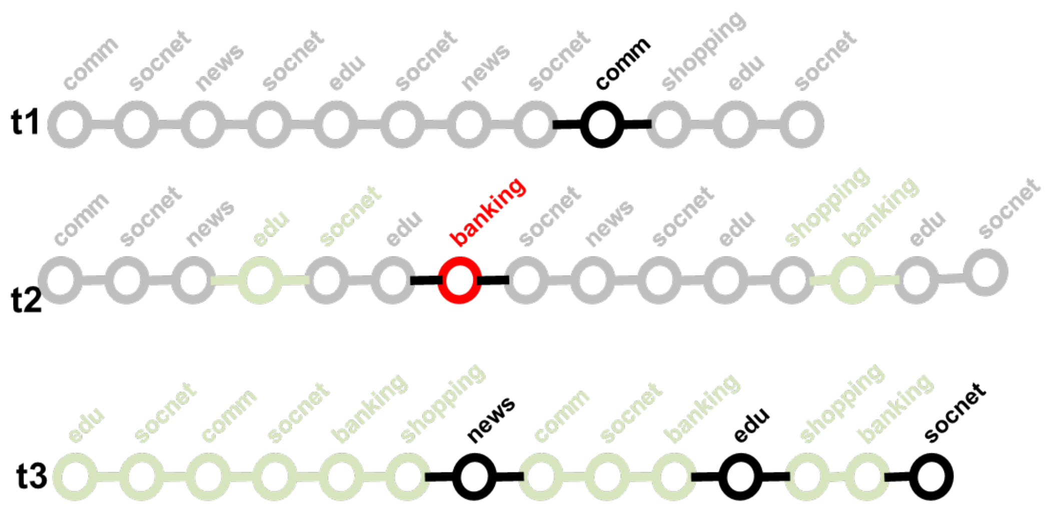


Figure 25. Timelines with six terminal set.

The introduction of the `comm` terminal also changed the bigram vocabulary:

- `shoppingbanking`
- `commsocnet`
- `socnetnews`
- `(commsocnet)banking`
- `(commsocnet)news`
- `socnetedu`
- `((commsocnet)banking)shopping`
- `socnet(commsocnet)banking`
- `edusocnet`

The resulting change is a decrease in the amount of activities explainable as part of a pattern, leaving five activities unattributed; most importantly though, the red activity was still included correctly left in the remaining activity set.

4.4 Summary

This chapter demonstrated reduction using SCFG parsing and structure inference in the association and individualization forensic processes, respectively. SCFG parsing to performed association by using the parsing sequences with known grammars and attributing the sequence to the grammar that had the highest most-probable parse likelihood. Individualization applied alignment and bigram-based inference to discover behavioral patterns that identify events as routine. By eliminating routine events from activity sequences, the examiner can focus on the remaining, unexplained events in the timelines.

V. Conclusions and Recommendations

Digital forensics examinations require significant manual effort to identify items of probative value from the ever-increasing volume of data in modern computing systems. This research proposes that Stochastic Context-Free Grammar (SCFG) knowledge representation can assist examiners in the association and individualization analysis processes on computer network traffic. SCFG is leveraged to provide context to the low-level data collected as evidence and to build behavior profiles. Upon discovering patterns, the analyst can begin the association or individualization process to answer criminal investigative questions. SCFG capabilities were demonstrated in performing association and individualization in reducing the suspect pool and reducing the volume of evidence to examine in a computer network traffic analysis use case.

Three contributions resulted from this research. First, domain characteristics suitable for SCFG representation were identified and a step-by-step approach to adapt SCFG onto novel domains was developed, enabling the PCAP to SCFG terminal process that translating low-level networking capture file into user activity sequences.

Second, performing the association process on user activity sequences required a set of known behavioral profiles. This necessitated a way to compare the different profiles, that led to the development of a novel iterative graph-based method of identifying similarities in context-free grammars, enabling comparisons between behavior patterns represented as grammars.

Third, SCFG parsing and structure inference performed association and individualization forensic processes to reduce the suspect pool or to reduce activity sequences to events of probative value. The results from these forensic processes answer investigative questions in a manner conveyable to a non-technical audience. Parsing produces a quantitative measure associating the most likely origin. Structure inference explained pattern events so that examination can focus on unattributed events.

5.1 Results Summary

PCAP to SCFG terminal processing is possible through Wireshark ordering of TCP streams, the IP meta data and organic keywords, and web categorization databases. These factors enable a discrete and causal sequence compatible with SCFG knowledge representation. The cross-disciplinary examination of SCFG applications identified SCFG-compatible domain characteristics and domain adaptations that other researchers can leverage to apply SCFG to other domains.

Association relied on existing profiles as grammars and the stochastic parameters of the production rules in the profiles. Parsing produced a comparative quantitative measure that enabled comparison between all the profiles in the set and association to the originating profile. The grammar comparison methodology developed as part of the experimental setup for this process has additional applications in NLP translation and computer language compiler interoperability analysis.

The alignment and bigram-based structure inference learner explained away the majority of activities in the set of activity sequences under examination, while not eliminating the anomalous user activity. The degree of reduction is sensitive to the PCAP to SCFG terminal process. The structure inference learning algorithms are domain independent and may have applicability to other domains.

5.2 Recommendations for Future Research

This section presents avenues of future research. The first recommendation addresses the issue of turning PCAPS into timelines. The next two recommendations are extensions of the association and individualization processes, based off topics from the discussion section for each process from Chapter 4. The association process may be extended to identify completed network attacks, which requires low-level signatures as expert defined production rules. The individualization application extends the grammar inference algorithms to produce test data that retains the behavior pattern of the users by first discovering patterns from user recorded computer network traffic. The final future research recommendation is transfer learning of SCFG stochastic parameters.

5.2.1 PCAP processing.

The PCAP to URL timeline was a manual and time consuming process. Recent efforts such as [87, 88] attempt to automate the process. Additional issues may also complicate timeline construction. Networks that use caching require different techniques for classification for association to a terminal. Dynamic IP addresses makes it difficult to determine whether different activities belong in a single individualization scenario. Range queries, which hides user queries with random dummy queries, adds significant noise to the timelines.

5.2.2 Association for Flexible Signatures.

The association sub-section 4.1.4 discusses the potential for parsing to identify the modus operandi of an attack from computer network traffic captures. This extends the PCAP processing future work to include recognition beyond URL categorization. The advantage to using SCFG is the flexibility to include additional low-level recognitions of these activities without impacting the rest of the knowledge structure. In addition, SCFG has advantages over just signature-based detection because SCFG parsing puts each phase of an attack into context of the entire five-phase attack, so incomplete attacks do not get flagged.

5.2.3 Data generation.

Despite obfuscating IP information, the information within search queries and visited sites may themselves reveal personally identifying information [71] studies and why studies that collect their own data cannot freely share their test data. The lack of a common data set is one challenge in performing network behavior analysis research, because cross comparison studies are difficult. From the privacy perspective, inferring an SCFG grammar to mimicking real user behavior by creating new data from capture data. This process creates a gap between the recorded data and user behavior so that the user tasks can be fulfilled using non-sensitive means. For instance, if a data capture records a user visiting www.facebook.com/specific_username, that event is abstracted as a `social` event in the grammar. When actually performing the `social` terminal, `social` may be satisfied by any site categorized as `social` and without reference to any specific username. Because the translation from categories into actual sites, this effort requires additional sensitivity analysis regarding terminal selection and noise, as discussed in sub-sections 4.3.1and 4.3.3. Precedence for SCFG in test data generation is described in Appendix B.

5.2.4 Transfer learning.

Transfer learning is the "ability of a system to recognize and apply knowledge and skills learned in previous tasks to novel tasks in new domains [89]." Inside-outside [90] is the machine learning algorithm typically used to calculate the stochastic parameters for each production in an SCFG. Because inside-outside's computational complexity is cubically driven by the number of productions and the number of inputs sequences, using the graph-based similarity methods may identify similar terminals across grammars and transfer the stochastic parameters matched terminals to reduce the cost of parameter learning.

Appendix A. SCFG Comparison

One of the reasons to use SCFG knowledge representation is to compactly represent large amounts of data as a set of rules governing the order of symbols. The comparison method also has application CFG problems such as language translation or identifying compatibility between compilers requires the ability to compare grammars.

Current methods of comparing CFGs occur at the rule-to-rule level or at the output-to-output level, which is a computationally impractical or undecidable problem. This paper presents a CFG comparison method that measures grammar similarity by identifying the structural symbol similarity between grammars. The presented method first produces a graph representation of the CFGs where nodes represent grammar symbols. Then, a graph node-matching algorithm produces a node similarity matrix between nodes, identifying nodes, and therefore CFG symbols that are most similar. The symbol matching then enables the CFG comparison of symbol connectivity and causalities, which measures the subset of similar symbol patterns between the two grammars. Results on several benchmark problems show that this method produces results in polynomial time and overcomes limitations in rule-to-rule and output-to-output grammar-based comparisons.

A.1 Reasons to Compare CFGs

There are multiple reasons to compare CFGs. First, CFGs represent large amounts of data. Making comparisons at the representation level avoids making more numerous output-to-output comparisons, which reduces the utility of a compact representation in the first place. Second, CFG comparisons show the existence of common patterns between the two sets of represented data. Language translation is an NLP application that exhibits these two elements of CFG comparison; it is not possible to enumerate and compare every sentence between two languages and translation benefits from understanding common sentence structure patterns [28]. Finally, grammar comparisons reveal potential incompatibilities across different grammars on the same data. An application of this scenario occurs in computer languages, where comparing grammars from different parsers for the same programming language will indicate varying acceptance levels over an identical piece of code [29].

Grammar-based methods evaluate grammar similarity using rule-to-rule comparisons [32, 33] or output-to-output comparisons [30, 31]. However, these approaches are computationally in exponential time [32] or undecidable [33–35]. The related work in this appendix provides a brief survey on different grammar-based similarity measurement concepts. Like CFGs, graphs represent vast amounts of data or highly-dimensional data in a compact matter [74]. Graphical models also describe logical structure in many real-world domains such as social networks, web addresses, and biology [77]. Algorithms that leverage graph structures often reveal useful information not obvious in its original data form. Graph comparisons methods include edit distance calculations, graph feature extraction, and iterative matching methods [77]. Edit distance approaches are also exponentially complex. Graph feature extraction methods are computationally fast, but are very sensitive to the selected statistics and may not produce intuitive results. Therefore, this paper uses an iterative graph similarity algorithm.

We address the undecidability issue of grammar comparison methods by translating the grammar into a graph and leveraging the advantages of graph-based representation for comparison. The graph captures the connectivity relationship of the CFG production rules into a single summarized presentation. Translating CFGs to graphs is not unprecedented. Muggleton and Pahlavi [74] relate CFG to a stochastic automata, by translating the production rules into states and transitions. Gecse and Kovacs [75] provide another example of translating CFG into graphs, for the purpose of identifying grammar consistency, highlighting a pragmatic benefit of examining CFG in graphical form. For comparison purposes, the proposed method uses a translation similar to Gecse and Kovacs [75], which converts the symbols into states and the links represent a connection between symbols within a CFG production rule. A difference is that all CFG symbols are represented as nodes in the graph, not just the non-terminals. The graph node matching algorithm used in the proposed approach has an $O(n^3)$ complexity.

Applying a graph node-matching algorithm provides a measure of similarity and compares grammars by matching a symbol in one grammar to its closest approximation in another symbol based on each symbol's connectivity to other symbols. The node-matching enable comparisons regarding the causality of symbols in CFG notation.

Similarity is measured as a combination of likeness between symbols and comparison of common causal relationships between symbols, where an existence of a causal link in both grammars indicate similarity while differences in causal links indicate dissimilarity. Comparisons of benchmark grammars show the intuitiveness of the results based on the node-matching results.

A.2 Related Work

In Wu, et al.'s [28] paper on language translation, the authors represent Chinese and Taiwanese sign-language as probabilistic context-free grammars. Their work focused on transferring the likelihood of each production rule to rank translations. The experiment required a bilingual corpus, where the same text appeared in both languages, which is an output-to-output comparison on a small subset of the languages. If a bilingual corpus is not available, translation requires alternative approaches to identify common rules in both languages.

Fischer, et al. [29] also used an output-to-output grammar comparison approach to detect Java parser incompatibilities. Each grammar generated a test data set comprised of auto-generated code from the parser's CFG. They determined parser compatibility based on how much each parser accepted test data generated by a different parser. While they used an output-to-output based approach, they found that identifying non-terminal matching is useful in understanding grammar compatibility.

Rule-to-rule or grammar based similarity comparisons examine similarity from the perspective of structural equivalence [33, 35] and weak equivalence (coverage) [34, 35]. Structural equivalence performs comparisons by iterating over the production rules and determining whether symbols and their respective rules exist or in combination exist in both grammars. Paull and Unger [33] define two grammars as *structurally equivalent* if both grammars produce the same sentences using different production rules. Hunt, et al. [32] conjecture that structural equivalence is not polynomially-bounded.

An alternative grammar-based equivalence approach explores the concept of *weak equivalence*, which focuses only on equivalent coverage or output-to-output comparisons.

Two grammars are weakly equivalent if both grammars produce the same set of sentences. Unlike structural equivalence, weak equivalence is not concerned with how each grammar produces the same sentence, so weakly equivalent grammars do not necessarily preserve semantics. An application of weak equivalence is grammar compaction. Grammar compaction attempts to reduce the number of production rules within a single grammar by eliminating production rules that are parseable with other production rules, at the cost of some semantic information [35, 91]. However, there are no known algorithms to determine weak equivalency between grammars to pragmatically leverage this concept [34, 35].

To address the pragmatic issues with determining weak equivalence, Hunt and Rosenkrantz [92] approach similarity from the perspective of structural containment, or Reynolds covering, where one grammar is able to map production rules to create rules of the other grammar. They determined that finding Reynolds coverage between arbitrary grammars is an NP-complete problem, but polynomial-time algorithms exist for restricted grammars [92]. Soisalon-Soinen and Wood [31] examine a different covering relationship, *undercover*, based on the produced sentences. However, they also found that determining undercover relationship between two unrestricted CFGs is also an undecidable problem.

Tree-based comparisons are an intuitive transition to compare acyclic grammars in a graphical model. The parsing operation of a grammar resembles a tree-like hierarchy where the S is the root node, the internal nodes are V_N, and all the leaf nodes are V_T. Summars-Stay, et al. [93] applied tree-based operations to examine grammar similarity where the similarity measure is the cost of transforming one tree into the other. The cost is calculated as the sum of the cost of insertion, deletion, or relabeling operations [94] . For cross comparisons, Bulter [95] normalizes cost to the number of nodes in the larger tree. Calculating tree edit-distance is $O(n^3)$ [96]. To reduce the computational cost, Rui, et al. [97] expanded on the tree edit distance approach by transforming the trees into a binary vector representation that preserves the semantic relationship between nodes, but enables a linear complexity search that identifies the lower-bound relationship between trees. Knowing the lower-bound narrows the search space by filtering candidates that do not meet the lower-bound.

Recursive (cyclic) CFGs are not representable as a tree. Therefore, more general graphs are required to represent a broader scope of CFGs. Rosenkrantz and Hunt [30] related grammar isomorphism to graph isomorphism, though their finding applies only to regular grammars, which is less expressive than CFGs. Gecse and Kovacs [75] provide a CFG graphical representation as directed graphs. Their focus was in identifying grammar consistency rather than performing comparisons between grammars, but their work provides an example of the pragmatic benefits of examining CFG in graph form.

Once a CFG is in graph form, there are three approaches to measuring graph similarity: graph isomorphism, feature extraction, and iterative. The graph isomorphism approach applies edit distance operations to make one graph isomorphic to another [77]. Similarity is therefore, a quantitative measurement of the number of addition, subtraction, or substitution operations on edges and nodes. Algorithms determining graph isomorphism are exponential and thus impractical on large-scale graphs. In contrast, the feature extraction approach attempts to compare graph features such as degree distribution, diameter, or eigenvalues [77]. Algorithms to calculate these features scale well as graph size increases, but are sensitive to feature selection. Improper selection may yield unintuitive results [77]. Iterative methods approach similarity as a node-by-node comparison of their neighborhood and edges [36, 98].

Graph-based approaches also introduce the concept of sub-graph matching, where one graph may be a smaller portion of another. This is applicable to CFGs as one grammar may be a subset of another, analogous to the grammar-based coverage concepts. Sub-graph matching typically requires node mapping, which is analogous to comparing grammars that do not share the same $V_T \cup V_N$ set or labels.

A.3 Example

Upon increasing the complexity of the grammars, straight-forward rule-to-rule evaluation becomes less clear. Figure 26 shows a grammar, G_A, its graphical representation, and a list of the causalities from the grammar production rules and Figure 27 shows a comparison between G_A against G_C and G_B.

The grammar G_B is G_A converted into *Chomsky Normal Form* (CNF). In CNF, all production rules are in the form $A \rightarrow B\ C$ or $A \rightarrow a$, restricting the RHS of a production rule to two non-terminals (upper-case letters) or a single terminal (lower-case letters). All CFGs have an equivalent CFG representation [39]. By definition, the CNF version of the grammar is weakly equivalent to the original, but RHS restrictions cause structural changes, evident in the graphical representation. G_B includes additional nodes and lines not in G_A, which also complicate node matching.

The grammar G_C has an additional nonterminal symbol than G_A, but less than G_B, and can produce sentences not parseable by G_A. From the production rules, G_C also exhibits a causality relationship between a non-terminal and terminal symbol, A and a, that mimic the causality relationships in G_A between A and b, not exhibited in G_B. Once the node matches are applied to the causality lists in G_B and G_C, the causality lists indicate that G_B has causal pairs in common with G_A, despite a larger set of symbols, in contrast to G_C.

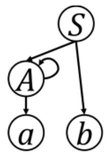
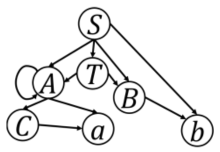
G_A				G_A Graph	G_A Causalities
V_T:	a, b				$A > b$
V_N:	S, A				$b > A$
S:	S				$A > a$
P:					
	S	$\rightarrow$	$A\ b\ A$		
	S	$\rightarrow$	$A\ b$		
	S	$\rightarrow$	$b\ A$		
	S	$\rightarrow$	b		
	A	$\rightarrow$	$A\ a$		
	A	$\rightarrow$	a		

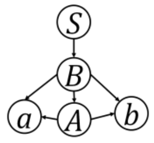
Figure 26. Grammar G_A, the basis for comparison.

A.4 Summary

Comparing context-free grammars (CFGs) has demonstrated use in applications such as language translation in Natural Language Processing (NLP), compiler compatibility, and activity recognition. CFGs compactly represent vast amounts of data. Making comparisons at the data representation level avoids computationally costly output-to-output comparisons and reveals common patterns between the sets of data. Grammar-based similarity concepts use rule-to-rule or output-to-output comparisons which are also computationally impractical or undecidable.

G_B				G_C			
V_T:	a, b			V_T:	a, b		
V_N:	S, T, A, B, C			V_N:	S,A,B		
S:	S			S:	S		
P:				P:			
	S	$\rightarrow$	$T\ A$		S	$\rightarrow$	B
	S	$\rightarrow$	$B\ A$		A	$\rightarrow$	a
	S	$\rightarrow$	$A\ B$		B	$\rightarrow$	$A\ a$
	S	$\rightarrow$	b		B	$\rightarrow$	$a\ A$
	T	$\rightarrow$	$A\ B$		B	$\rightarrow$	b
	A	$\rightarrow$	$A\ C$				
	A	$\rightarrow$	a				
	B	$\rightarrow$	b				
	C	$\rightarrow$	a				





X	S_A	A_A	a_A	b_A
S_B	0.13	0.14	0.00	0.00
T_B	**0.04**	0.21	0.00	0.00
B_B	0.00	0.05	0.02	**0.01**
A_B	0.14	**0.85**	0.07	0.02
C_B	0.02	0.13	0.06	0.01
a_B	0.04	0.34	**0.09**	0.01
b_B	0.00	0.02	0.01	0.01

X	S_A	A_A	a_A	b_A
S_C	0.09	0.09	0.00	0.00
A_C	**0.09**	0.43	0.08	0.02
B_C	0.26	**0.72**	0.01	0.01
a_C	0.00	0.25	**0.14**	0.04
b_C	0.00	0.25	0.14	**0.04**

G_B Causalities		*Remapped G_B nodes to G_A*		*G_C Causalities*		*Remapped G_C nodes to G_A*
$A > B$	→	$A > b$	*in G_A*	A > a	→	S > a
$B > A$	→	$b > A$	*in G_A*	a > A	→	a < S
$T > A$	→	$S > A$				
$A > C$	→	$A > C$				

Figure 27. Two grammars, G_B and G_C, compared against G_A. G_B is G_A in Chomsky Normal Form. G_C is covered by G_A and though it has a symbol set more similar to G_A, G_C is less similar to G_A than G_B as indicated by remapped causal relationships.

We translate CFG into graphs to leverage known polynomial-time similarity algorithms. Turning CFGs into graphs collapses the connectivity relationship between symbols from all the production rules into one summarized presentation for efficient comparison. We selected a polynomial complexity graph node-matching algorithm that produces a likelihood matrix that matches grammar symbols across the grammars under comparison. Similarity is based on the likeness between symbols across two grammars and the existence of common causal links between symbols in each grammar. Benchmark programs produced intuitive results in comparison to known grammar concepts.

The proposed approach may yield promising results in transfer learning, which commonly includes taxonomies translatable to CFG. Identifying similarities reduces the dependence on labeled data and may achieve better learning results.

The proposed method does not leverage stochastic information found in stochastic context-free grammars (SCFG). Additional work may extend the proposed method to incorporate the stochastic parameters to measure SCFG similarity.

Appendix B. SCFG for Test Data Generation

Vishwanath and Vahdat [99] developed Swing, a network traffic generator to produce usage traces that reflect the characteristics of template traces at the packet level. They adopted a structural model determine the type of traffic flow, which included a user category that determined the application used and a session category that determined the higher-level task carried out by the user. Thus, Swing reproduces packet level characteristics, such as byte and packet burstiness, by using notions of the higher-level user and session concepts. Because of Swing's focus at the packet level, most of their work discusses details below the session layer.

Chinchilla, et al. [100] proposed a traffic emulator, trafgen, to model traffic at the transport protocol level. They outlined user behavior modeling as combinations of Markov (memory-less), Petri Nets (preconditional), Hierarchical, and psychological/sociological transitions. Users are denoted as specific IP addresses.

Simpson, et al. [101] collected user network data from approximately 1,700 volunteers who downloaded and ran the NETI@home (NETwork Intelligence at home) client on their machines. Using the client allowed the authors to gain insight into end-user behavior in a network-independent manner. Capturing information at the user end also avoids the complexity of accounting for proxies and caches. The study captured behaviors based on TCP or UDP ports, user think time, consecutive contacts, and contact selection. The modeling approach did not attempt to categorize user type or specific user tasks.

Gold, et al. [102, 103] developed GOSMR (Goal-Directed Scenario Modeling Robots) in an attempt to add the element of goal-directed behaviors in network traffic generators. GOSMR focuses on the activity of the agents, particularly agent response to satisfy goals under network disruption. Each agent has stack of Actions. Actions are self-contained, though actions can put lower-level actions onto the stack to satisfy itself. A Behavior object suggests an action for the using the utility and payoffs to choose the action, an agent has an empty stack. GOSMR can also alter the payoffs and penalties to mimic certain conditions to force the agents to take different actions as a reflection of the changed conditions to satisfy the same goal. GOSMR also includes a partial-order search through plan space planner to provide an agent alternate actions to achieve the agent's goal in the event an attempted action fails. An advantage of their framework is that the set of agent behaviors is modular and can be added or inserted as needed, without modifying the overall architecture. In their experiments, the agents all act accordingly to the same set of rewards, so different agents tended to behave the same way, such as using faster responding services more.

Maurer [104] documented early experience of using "enhanced" CFGs to produce data to test code. His enhancement to CFGs includes using stochastic parameters, essentially SCFGs. To test specific branches, he applied selective substitution to ensure coverage. Empirically, he set the recursive productions to a smaller probability such that tests did not grow infinitely large.

Gecse and Kovacs [75] propose a method for evaluating the consistency of SCFG and transforming arbitrary SCFGs into consistent ones, with the specific emphasis on producing test data. Consistency is defined as the likelihood of terminating a sentence generation after a finite number of steps. Consistency is important for test data generation when no corpus is available to infer probability parameters that yields sufficient coverage and error tolerance.

Zhao, et al. [105] discover behavioral patterns in computer programs for the purpose of identifying common substructures to ease debugging and to also simulate program executions. Their approach used context-sensitive graph grammars and substructure compression, an MDL approach, to identify the substructure patterns.

Buehrer, et al. [106] examined the differences between automated traffic and human generated traffic based on behavioral patterns. Used legimitely, generating more realistic traffic patterns may help test networks. Used maliciously, this can also be used to garner click-through rates. From the aspect of test data generation, their work provides insight into how to distinguish generated versus empirical data. They were able to distinguish between the groups, but the classification was based primarily on activity rate and volume factors rather than the activities themselves.

Bibliography

[1] R. J. Walls, B. N. Levine, M. Liberatore, and C. Shields, “Effective digital forensics research is investigator-centric,” in *Proc. USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[2] F. Buchholz and C. Falk, “Design and Implementation of Zeitline: a Forensic Timeline Editor.,” *Digit. Forensic Res. Work.*, pp. 1–7, 2005.

[3] S. Esposito, “Analysis of Forensic Super Timelines,” Air Force Institute of Technology, 2012.

[4] P. Gladyshev and A. Patel, “Formalising event time bounding in digital investigations,” *Int. J. Digit. Evid.*, vol. 4, no. 2, pp. 1–14, 2005.

[5] J. Olsson and M. Boldt, “Computer forensic timeline visualization tool,” *Digit. Investig.*, vol. 6, no. SUPPL., pp. S78–S87, 2009.

[6] “Digital Evidence and Forensics,” 2010. [Online]. Available: http://www.nij.gov/topics/forensics/evidence/digital/Pages/welcome.aspx.

[7] E. S. Pilli, R. C. Joshi, and R. Niyogi, “Network forensic frameworks: Survey and research challenges,” *Digit. Investig.*, vol. 7, no. 1–2, pp. 14–27, 2010.

[8] M. Pollitt, “Applying traditional forensic taxonomy to digital forensics,” *IFIP Int. Fed. Inf. Process.*, vol. 285, pp. 17–26, 2008.

[9] H. C. Lee and E. M. Pagliaro, “Forensic Evidence and Crime Scene Investigation,” *J. Forensic Investig.*, vol. 1, no. 1, pp. 1–5, 2013.

[10] K. Inman and N. Rudin, “The origin of evidence,” *Forensic Sci. Int.*, vol. 126, no. 1, pp. 11–16, 2002.

[11] “Nitroba University Harassment Scenario,” 2010. [Online]. Available: http://digitalcorpora.org/corpora/scenarios/nitroba-university-harassment-scenario.

[12] C. Perera, S. Member, A. Zaslavsky, and P. Christen, “Context Aware Computing for The Internet of Things : A Survey,” *Commun. Surv. Tutorials, IEEE*, vol. 16, no. 1, pp. 414–454, 2014.

[13] C. W. Geib and M. Steedman, “On Natural Language Processing and Plan Recognition.,” in *International Joint Conferences on Artificial Intelligence*, 2007, pp. 1612–1617.

[14] N. Indurkya and F. J. Damerau, *Handbook of natural language processing*, 2nd ed. 2010.

[15] M. Olsen, R. Horton, and G. Roe, “Something borrowed: sequence alignment and the identification of similar passages in large text collections.,” *Digit. Stud. champ numérique*, vol. 2, no. 1, 2011.

[16] Y. A. Ivanov and A. F. Bobick, “Recognition of visual activities and interactions by stochastic parsing,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 852–872, 2000.

[17] D. Moore and I. Essa, “Recognizing multitasked activities from video using stochastic context-free grammar,” *AAAI/IAAI*, pp. 770–776, 2002.

[18] R. Nevatia, T. Zhao, and S. Hongeng, “Hierarchical language-based representation of events in video streams,” in *Computer Vision and Pattern Recognition Workshop, 2003. CVPRW’03.*, 2003, vol. 4, p. 39.

[19] K. M. Kitani, Y. Sato, and A. Sugimoto, “Recovering the basic structure of human activities from a video-based symbol string,” in *Motion and Video Computing, 2007*, 2007, p. 9.

[20] E. Rivas and S. R. Eddy, “The language of RNA: a formal grammar that includes pseudoknots,” *BMC Bioinformatics*, vol. 16, no. 4, pp. 334–340, 2000.

[21] D. Searls, “The language of genes,” *Nature*, vol. 420, no. 6912, pp. 211–217, 2002.

[22] Y. Sakakibara, “Grammatical inference in bioinformatics,” *Pattern Anal. Mach. Intell.*, vol. 27, no. 7, pp. 1051–1062, 2005.

[23] R. Giegerich, “Introduction to Stochastic Context Free Grammars,” in *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods.*, Humana Press, 2011, pp. 85–106.

[24] J. Anderson, P. Tataru, and J. Staines, “Evolving stochastic context--free grammars for RNA secondary structure prediction,” *BMC Bioinformatics*, vol. 13, no. 1, p. 78, 2012.

[25] C. W. Geib and R. P. Goldman, “A probabilistic plan recognition algorithm based on plan tree grammars,” *Artif. Intell.*, vol. 173, no. 11, pp. 1101–1132, Jul. 2009.

[26] C. Mu and Y. Li, “An intrusion response decision-making model based on hierarchical task network planning,” *Expert Syst. Appl.*, vol. 37, no. 3, pp. 2465–2472, Mar. 2010.

[27] N. Li, S. Kambhampati, and S. W. Yoon, “Learning Probabilistic Hierarchical Task Networks to Capture User Preferences.,” in *International Joint Conference on Artificial Intelligence*, 2009, pp. 1754–1759.

[28] C.-H. Wu, H.-Y. Su, Y.-H. Chiu, and C.-H. Lin, “Transfer-based statistical translation of Taiwanese sign language using PCFG,” *ACM Trans. Asian Lang. Inf. Process.*, vol. 6, no. 1, p. 1–es, Apr. 2007.

[29] B. Fischer, R. Lämmel, and V. Zaytsev, “Comparison of context-free grammars based on parsing generated test data,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6940 LNCS, pp. 324–343, 2012.

[30] D. J. Rosenkrantz and H. B. Hunt, “Testing for grammatical coverings,” *Theor. Comput. Sci.*, vol. 38, no. 3, pp. 323–341, 1985.

[31] E. Soisalon-Soininen and D. Wood, “On structural similarity of context-free grammars,” in *Mathematical Foundations of Computer Science*, J. Gruska and M. Chytil, Eds. Springer Berlin Heidelberg, 1981, pp. 491–498.

[32] H. B. Hunt III, D. J. Rosenkrantz, and T. G. Szymanski, “On the equivalence, containment, and covering problems for the regular and context-free languages,” *J. Comput. ...*, vol. 268, pp. 222–268, 1976.

[33] M. M. C. Paull and S. H. S. Unger, “Structural equivalence of context-free grammars,” *J. Comput. Syst. Sci.*, vol. 463, pp. 427–463, 1968.

[34] P. S. Landweber, “Decision Problems of Phrase-Structure Grammars,” *IEEE Trans. Electron. Comput.*, vol. EC-13, 1964.

[35] K. Taniguchi and T. Kasami, “Reduction of context-free grammars,” *Inf. Control*, vol. 108, no. September 1968, pp. 92–108, 1970.

[36] L. a. Zager and G. C. Verghese, “Graph similarity scoring and matching,” *Appl. Math. Lett.*, vol. 21, pp. 86–94, 2008.

[37] R. Davis, H. Shrobe, and P. Szolovits, “What Is a Knowledge Representation?,” *AI Magazine*, vol. 14, no. 1, pp. 17–33, 1993.

[38] M. Collins, “Probabilistic Context-Free Grammars (PCFGs).” 2013.

[39] C. D. Manning and H. Schutze, “Chapter 11. Probabilistic Context Free Grammars,” in *Foundations of Statistical Natural Language Processing*, 1999, pp. 381–404.

[40] D. H. Younger, “Recognition and Parsing of Context-Free Languages in Time n3,” *Inf. Control*, vol. 10, pp. 189–208, 1967.

[41] J. Earley, “An efficient context-free parsing algorithm,” *Communications of the ACM*, vol. 13. pp. 94–102, 1970.

[42] S. M. Shieber, “Evidence against the context-freeness of natural language,” *Linguist. Philos.*, pp. 333–343, 1985.

[43] R. Bod, “Using an annotated corpus as a stochastic grammar,” in *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, 1993, pp. 37–44.

[44] L. Lee, “Learning of context-free languages: A survey of the literature,” 1996.

[45] J. Reeder, P. Steffen, and R. Giegerich, “Effective ambiguity checking in biosequence analysis,” *BMC Bioinformatics*, vol. 6, no. 1, p. 153, 2005.

[46] V. O. Polyanovsky, M. A. Roytberg, and V. G. Tumanyan, “Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences.,” *Algorithms Mol. Biol.*, vol. 6, no. 1, p. 25, Jan. 2011.

[47] Z. Zhang, T. Tan, and K. Huang, “An extended grammar system for learning and recognizing complex visual events,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 33, no. 2, pp. 240–255, 2011.

[48] N. Li, W. Cushing, S. Kambhampati, and S. Yoon, “Learning User Plan Preferences Obfuscated by Feasibility Constraints.,” *Int. Conf. Autom. Plan. Sched.*, 2009.

[49] K. Erol, J. Hendler, and D. S. Nau, “HTN planning: Complexity and expressivity,” in *AAAI*, 1994, vol. 94, pp. 1123–1128.

[50] G. G. Chowdhury, “Natural language processing,” in *Annual Review of Information Science and Technology*, 2003.

[51] N. M. Luscombe, D. Greenbaum, and M. Gerstein, “What is bioinformatics ? An introduction and overview,” *Yearb. Med. Inform.*, pp. 83–100, 2001.

[52] F. Coste, “Tutorial on modelling biological sequences by grammatical inference: Bibliography,” *Engineering*, vol. 1, no. August, pp. 107–151, 2010.

[53] R. D. Dowell and S. R. Eddy, “Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction,” *BMC Bioinformatics*, vol. 5, no. 1, p. 71, 2004.

[54] M. S. Ryoo and J. K. Aggarwal, "Stochastic Representation and Recognition of High-Level Group Activities," *Int. J. Comput. Vis.*, vol. 93, no. 2, pp. 183–200, Jun. 2010.

[55] M. S. Ryoo and J. K. Aggarwal, "Recognition of composite human activities through context-free grammar based representation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006, pp. 1709–1718.

[56] T. Zimmerman and S. Kambhampati, "Learning-assisted automated planning: Looking back, taking stock, going forward," *AI Magazine*, vol. 24, no. 2, p. 73, 2003.

[57] D. V Pynadath and M. P. Wellman, "Probabilistic state-dependent grammars for plan recognition," in *Uncertainty in Artificial Intelligence Proceedings*, 2000, pp. 507–514.

[58] C. W. Geib and R. P. Goldman, "Recognizing Plans with Loops Represented in a Lexicalized Grammar.," in *AAAI*, 2011.

[59] N. Li, W. Cushing, S. Kambhampati, and S. Yoon, "Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 2, pp. 1–34, 2014.

[60] M. E. DeYoung, "Dynamic protocol reverse engineering: A grammatical inference approach," Thesis, Air Force Institute of Technology, 2008.

[61] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," in *18th Working Conference on Reverse Engineering (WCRE), 2011*, 2011, pp. 169–178.

[62] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo, "Stochastic protocol modeling for anomaly based network intrusion detection," in *First IEEE International Workshop on Information Assurance, 2003. IWIAS 2003.*, 2003, pp. 3–12.

[63] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the Inside-Outside algorithm," *Comput. Speech Lang.*, vol. 4, no. 1, pp. 35–56, Jan. 1990.

[64] J. J. Horning, "A Study of Grammatical Inference," Dissertation, Stanford University, 1969.

[65] A. D'Ulizia, F. Ferri, and P. Grifoni, "A survey of grammatical inference methods for natural language learning," *Artif. Intell. Rev.*, vol. 36, no. 1, pp. 1–27, Jan. 2011.

[66] G. Combs, "Wireshark." 2015.

[67] H. Kim, K. Claffy, and M. Fomenkov, "Internet traffic classification demystified: myths, caveats, and the best practices," in *ACM Conference on emerging Networking EXperiments and Technologies*, 2008.

[68] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surv. Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[69] A. Barsamian, V. Berk, and J. Murphy, "Identifying Network Users Using Flow-Based Behavioral Fingerprinting." 2013.

[70] Y. Yang, "Web user behavioral profiling for user identification," *Decis. Support Syst.*, vol. 49, no. 3, pp. 261–271, 2010.

[71] C. Banse, D. Herrmann, and H. Federrath, "Tracking users on the Internet with behavioral patterns: Evaluation of its practical feasibility," *IFIP Adv. Inf. Commun. Technol.*, vol. 376 AICT, pp. 235–248, 2012.

[72] W. Mao, J. Gratch, and X. Li, "Probabilistic plan inference for group behavior prediction," *IEEE Intell. Syst.*, vol. 27, no. 4, pp. 27–36, 2012.

[73] M. Rogers, "The role of criminal profiling in the computer forensics process," *Comput. Secur.*, vol. 22, no. 4, pp. 292–298, 2003.

[74] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[75] R. Gecse and A. Kovács, "Consistency of stochastic context-free grammars," *Math. Comput. Model.*, vol. 52, no. 3–4, pp. 490–500, 2010.

[76] Yi Cao (Cranfield University), "Munkres (Hungarian) Algorithm for Linear Assignment Problem." 2011.

[77] D. Koutra, a Parikh, a Ramdas, and J. Xiang, "Algorithms for Graph Similarity and Subgraph Matching," 2011.

[78] A. J. Laub, *Matrix Analysis for Scientists and Engineers*. 2005.

[79] J. Edmonds and R. M. Karp, “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems,” *J. ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[80] T. Broeders, “Forensic evidence and international courts and tribu nals:,” pp. 1–10, 2003.

[81] H. K. Peng, P. Wu, J. Zhu, and J. Y. Zhang, “Helix: Unsupervised grammar induction for structured activity recognition,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 1194–1199, 2011.

[82] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins.,” *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[83] N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, vol. 101, no. 474. 2004.

[84] C. G. Nevill-Manning and I. H. Witten, “Identifying Hierarchical Structure in Sequences: A linear-time algorithm,” vol. 7, pp. 67–82, 1997.

[85] E. Skoudis and T. Liston, *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Pearson Education, 2005.

[86] V. Gorodetski and I. Kotenko, “Attacks against computer network: Formal grammar-based framework and simulation tool,” in *Recent Advances in Intrusion Detection*, 2002, pp. 219–238.

[87] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin, “ReSurf: Reconstructing Web-Surfing Activity From Network Traffic,” *IFIP Netw. Conf.*, pp. 1–9, 2013.

[88] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms, “ClickMiner : Towards Forensic Reconstruction of User-Browser Interactions from Network Traces Categories and Subject Descriptors,” *Proc. 2014 ACM SIGSAC Conf. Comput. Commun. Secur. ACM*, pp. 1244–1255, 2014.

[89] S. J. Pan and Q. Yang, “A Survey on Transfer Learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[90] M. Johnson, “Open-source software by Mark Johnson,” 2007. [Online]. Available: http://web.science.mq.edu.au/~mjohnson/Software.htm.

[91] M. Hepple and J. Van Genabith, “Experiments in Structure Preserving Grammar Compaction,” no. PTB II, 1999.

[92] H. B. Hunt and D. J. Rosenkrantz, “Efficient algorithms for structural similarity of grammars,” in *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL ’80*, 1980, pp. 213–219.

[93] D. Summers-Stay, C. L. Teo, Y. Yang, C. Fermuller, and Yiannis Aloimonos, “Using a Minimal Action Grammar for Activity Understanding in the Real World,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 4104–4111, 2012.

[94] K. Zhang and D. Shasha, “Simple Fast Algorithms for the Editing Distance between Trees and Related Problems,” *SIAM J. Comput.*, vol. 18, no. 6, pp. 1245–1262, 1989.

[95] D. Buttler, “A short survey of document structure similarity algorithms,” in *International Conference on Internet Computing*, 2004, pp. 3–9.

[96] P. Bille, “A survey on tree edit distance and related problems,” *Theor. Comput. Sci.*, pp. 1–27, 2005.

[97] R. Yang, P. Kalnis, and A. K. H. Tung, “Similarity evaluation on tree-structured data,” *Proc. 2005 ACM SIGMOD Int. Conf. Manag. data SIGMOD 05*, p. 754, 2005.

[98] G. Kollias, “Fast parallel algorithms for graph matching problems,” *Comput. Math. with Appl.*, vol. 36, p. 123, 1998.

[99] K. V. Vishwanath and A. Vahdat, “Swing: Realistic and responsive network traffic generation,” *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 712–725, 2009.

[100] R. Chinchilla, J. Hoag, D. Koonce, H. Kruse, S. Osterman, and Y. Wang, “Characterization of Internet Traffic and User Classification: Foundations for the Next Generation of Network Emulation,” 2002.

[101] C. R. Simpson, D. Reddy, and G. F. Riley, “Empirical Models of End-User Network Behavior from NETI@home Data Analysis,” *Simulation*, vol. 84, no. 10–11, pp. 557–571, 2008.

[102] K. Gold, Z. J. Weber, B. Priest, J. Ziegler, K. Sittig, and W. W. Streilein, “Modeling How Thinking About the Past and Future Impacts Network Traffic with the GOSMR Architecture Categories and Subject Descriptors,” *Proc. AAMAS*, 2013.

[103] K. Gold and W. Street, “Learning to Efficiently Pursue Communication Goals on the Web with the GOSMR Architecture,” pp. 1227–1233, 2013.

[104] P. M. Maurer, “Generating test data with enhanced context-free grammars,” *IEEE Softw.*, vol. 7, no. 4, 1990.

[105] C. Zhao, K. Ates, J. Kong, and K. Zhang, “Discovering program’s behavioral patterns by inferring graph-grammars from execution traces,” *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*, vol. 2, pp. 395–402, 2008.

[106] G. Buehrer, J. W. Stokes, and K. Chellapilla, “A large-scale study of automated web search traffic,” *Proc. 4th Int. Work. Advers. Inf. Retr. web - AIRWeb ’08*, p. 1, 2008.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 074-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* 17-09-2015	**2. REPORT TYPE** Doctoral Dissertation		**3. DATES COVERED** *(From – To)* March 2013 – Sep 2015
TITLE AND SUBTITLE Network Analysis with Stochastic Grammars		**5a. CONTRACT NUMBER**	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Lin, Alan C., Maj, USAF		**5d. PROJECT NUMBER**	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865			**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-DS-15-S-014
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally left blank			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **DISTRUBTION STATEMENT A.** APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.			

14. ABSTRACT

Digital forensics requires significant manual effort to identify items of evidentiary interest from the ever-increasing volume of data in modern computing systems. Digital forensic examiners mentally extract insights from unstructured sequences of events. To help examiners prioritize and extract useful information from low-level network data, this research hypothesizes using Stochastic Context-Free Grammar (SCFG) knowledge representation to perform behavior analysis in network forensic examination. Results demonstrated that SCFG produced a qualitative measure to compare behavior profiles and associate the likely source. In addition, the development of two grammar inference methods identified behavior patterns in network data, to narrow the search space and focus examination on events unexplained by the behavior patterns.

15. SUBJECT TERMS
Data mining, Stochastic Grammar, Pattern Discovery

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Gilbert L. Peterson, AFIT/ENG
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	123	(937)255-6565,ext4281 gilbert.peterson@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18