Genetic Programming for Password Cracking

Phase One: Grammar Induction

Thulani Mashiane
School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal
Pietermaritzburg, South Africa
Council of Scientific and Industrial Research
Pretoria, South Africa
tmashiane@ukzn.co.za

Nelishia Pillay
School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal
Pietermaritzburg, South Africa
Pillayn32@ukzn.ac.za

***Abstract*— *Password* cracking is the term commonly used to describe the illegal action of gaining access to clear text versions of user passwords. Hackers are notorious for stealing encrypted passwords and cracking them. The same action of password cracking can be used by system administrators to protect their systems from weak user passwords. By applying a password cracker to user passwords, weak or easy to crack passwords can be identified. Through the design of a password cracker system administrators can prevent weak passwords from being saved onto their systems. Users can also be made aware of the strength of the passwords they are currently employing. A manner in which password cracking can be made more effective is to produce a few guess words with a high probability of cracking a large number of passwords. Research has revealed the successful use of *grammars* to generate effective password guess words. In order to generate password grammars, genetic programming is applied to grammar induction for the purpose of inducing grammars that will be used as input to a password cracking tool. To achieve this goal the current paper looks at the performance of genetic programming in the induction of regular and context-free languages. The results of the experiments conducted are promising, with the genetic programming algorithm managing to induce twenty three of the twenty six context-free languages it was tested on. The value of this paper lies in the evaluation of the genetic programming technique for grammar induction. The output of the research will be used to build a genetic programming system which can evolve grammars to generate password guess words to crack user created passwords.**

Keywords*— *Genetic programming; grammar induction; password cracking

I. INTRODUCTION

A password is a secret chain of characters (alphabet letters, numbers or special characters) which is used as a key to gain entry into a system. A system assumes that a user that can produce the correct password has authorised access to it. The aim of password implementation is for authorised users to gain easy access to the system by producing the password, but impossible or at least very difficult for unauthorised users to produce the password. Different password implementations exist.Some are more technologically advanced such as biometric or graphical . The most basic text based. While other forms of password implementation are seen as more secure, text based passwords are still heavily used to protect resources such as funds in a bank, or to protect user privacy such as conversations on social network sites. In order to assign a password, there are two main approaches that are followed. The first approach is allowing users to create passwords for themselves. In the second approach the administrator creates the password and assigns it to the user.

In most cases password creation is left in the hands of users. It is common knowledge in the security domain that human users are the weakest link in many security systems. This is also true in the creation of text based passwords. The majority of users create weak passwords that are easy to guess. This can be attributed to users following a routine when completing tasks. These routines make the passwords created by users predictable and a security risk.

Researchers have used different techniques to search for patterns in user passwords. Using 14 000 password entries Bishop and Klein [1] searched for the commonly known patterns such as word length, dictionary words and keyboard patterns. Jakobsson and Dhiman [2] built a parser to uncover the internal structure of passwords. The research takes a two-phased approach ; the first phase was a search for components and the second focused on deriving rules that join the different components. Weir, Aggarwal, de Medeiros and Glodek [3] build a computer program that takes in as input probabilistic context-free grammars to create a password cracker. The structures of the grammars are essentially patterns that are derived for actual user passwords. The research presented in this paper would like to build on this work by applying genetic programming to the induction of grammars which will later be used for password cracking. The languages studied in the current work are a set of benchmark regular languages as well as a set of standard context-free grammar languages that have previously been used in studying grammar induction.

The breakdown for the rest of the paper is as follows: Section II gives an introduction to grammars and explains their relevance to password guess words. Section III gives an explanation of genetic programming. Section IV motivates by presenting previous research, why genetic programming can be used to evolve grammar structures. Section V presents the genetic programming algorithm used in the study. Section VI

describes the experiment setup. Section VII presents the results from the study. Section VIII provides an analysis of the results as well as a comparison to previous research. The paper is concluded in section IX. Future work is then presented in section X.

II. Grammars and Passwords

A grammar is a set of symbols, variables and rules. The rules of a grammar govern how the symbols and variables can be used in the generation of a language. Grammars for natural languages have their origin in the field of linguistics. Recently however, computer science researchers have been investigating how grammars can be applied in domains such as pattern recognition, image processing, and video processing. Grammars have also been used in the field of password cracking.

Formally a grammar is defined by the quadruple $G = (V, \Sigma, S, P)$ [6, 7]. V represents a finite set of non-terminal symbols. Σ is the alphabet of the grammar which consists of a set of terminal symbols. S is the start symbol and P represents the finite set of production rules in the form $\alpha \rightarrow \beta$ where $\alpha, \beta \in (V \cup \Sigma)$ and $\alpha \neq \varepsilon$. P represents the rules which are used to create the strings in a language [6, 7]. Grammar induction is defined as the challenge of creating a grammar from a given set of positive and negative samples. Among other approaches, evolutionary algorithms have been applied to the induction of regular and context-free grammars. The next section will discuss genetic programming, an evolutionary algorithm that has been previously applied to grammar induction, as well as previous research that has been conducted in the field.

User created passwords are created by using different characters which are combined in different ways or permutations. Through a survey of literature, it is found that there exists two general categories in password patterns; character patterns and word patterns. Character patterns are passwords or part of passwords that result from using one or more characters independently, or characters that do not form part of a word. Patterns that fall under this category are singular (a single alphabet character), repeated character (the same repeated character), increments (a repeated character of alphabet letters or numbers) and keyboard patterns (patterns which are derived through the order which they are presented on the keyboard). Word patterns are a result of mangling rules that are applied to password dictionary words. The patterns that fall under this category are capitalization (simply capitalizing every letter in the word), concatenation (placing one or more words together), l33t (leet) substitutions (replacing letters in dictionary words with numbers or symbols that resemble that letter), inverse and palindrome.

Grammars can be used to capture the structures which are found in user created passwords. For example the palindrome pattern can be represented by the grammar $G = (V, \Sigma, S, P)$ where :

$V = \{S, B\}$; $\Sigma = \{a, b\}$; $S = \{S\}$; $P = \{ S \rightarrow a\ S\ a ; S \rightarrow b\ S\ b; S \rightarrow a ; S \rightarrow b \}$

The palindrome ‘baabaab’ can then be constructed by using the grammar productions.

The next section gives an explanation of genetic programming.

III. Genetic Programming

The skill of writing computer programs is seen as difficult to acquire. Good programmers must have programming skills as well as some domain knowledge of the problem in order to build a solution. A programmer must show attention to detail as a single error, whether syntactical or logical, can cause an entire system to fail. This is one of the reasons that Computer Scientists have looked to alternative methods in building computer programs. Genetic programming is a technique of automatically evolving computer programs. Inspired by Darwin’s principle of evolution, genetic programming evolves a randomly generated population of programs towards a desired solution program [4].

A generational genetic programming run begins with creating an initial population of randomly generated programs. The individuals are then evaluated to find the individuals in the population that are closer to the solution program than others. A selection method is then used to select the individuals that will be used as parents. The next generation is created by applying genetic operators, usually reproduction, mutation and crossover to the parents. Once a new population has been created, the algorithm iterates until a solution to the problem is found or a stopping criterion is met. The complete process, starting with the initial population and iteratively refining this population over a number of generations, is known as a genetic programming run [4, 5]. Fig. 1. shows a diagram of a generational genetic programming run.

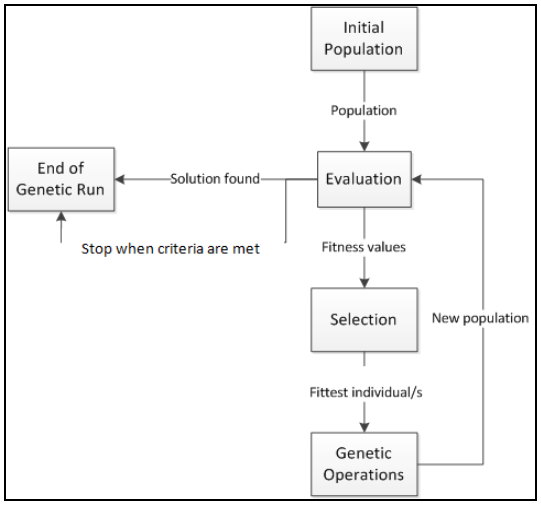

Fig. 1. Steps of a Generational Genetic Program Run

IV. Genetic Programming for Gramar induction

This section will present previous research using genetic programming for grammar induction.

Javed [6] implemented genetic programming for the induction of context-free grammars. The context-free grammars represent domain specific languages (DSL). Grammar representation was in Backus Naur Form. The initial

population was created by productions from a correct grammar for the language. One point crossover and mutation were implemented. These genetic operators were not sufficient for the induction of context-free grammars. Therefore three additional genetic operators were designed. These include the option operator, iteration + and iteration* operators. Option allows the production rule to choose between two productions. Iteration* allows a production rule to call a value on the left hand side of a production zero or more times. Iteration+ allows the production rule to call a value on the left hand side of a production at least once. The fitness measure computed the number of grammars that were correctly passed, the closeness of the inferred grammars to the correct grammar and lastly the scalability of the grammar. With the addition of the helper genetic operators the genetic programming system was able to induce small sized DSL context-free grammars.

In [7] grammar induction was revisited by Javed [7] with the use of genetic programming. The alterations to the previous system were in the representation of the grammars. The grammars were now represented using derivation trees and syntax graphs. A new helper operator called the delete operator was implemented. This operator deletes a production rule from a grammar. The genetic programming system was successful in evolving grammars for languages which required a small grammar structure. Larger grammar structures could not be induced. The helper operators were an important element in this study. They allowed the properties of context-free grammars to be represented in full. Properties such as allowing for recursion are captured using the iteration operators. An more suitable representation of a context-free grammar can capture the described properties of a grammar. This topic will be elaborated upon in the discussion of the current systems grammar representation.

Javed, Barrett, Matej Marajan and Alan [8] built on the work in [6] and [7]. The genetic programming system implemented is similar to [7]. The researchers suggested that the right hand side of the grammars be treated as a collection of regular sets. These regular sets are connected by union, concatenation and recursion. It was anticipated that the genetic programming system would find the small regular grammars of the big grammar and join them using concatenation, union or recursion. Again, what the researchers tried to capture in their suggestion is the properties or capabilities of context-free grammars. This observation implies that there are missing properties in grammar induction that prevents a solution from being evolved. This can be linked back to Koza's requirements for a genetic programming system. Koza [4] emphasised that the terminal set, function set, fitness function and problem representation must be implemented so that it is possible to evolve a solution. If the problem domain is not fully represented, the genetic program cannot evolve a solution.

Korkmaz and Üçoluk [11] also did some work on this topic. The grammars were represented as a structured tree. The fitness function used took into consideration both the number of correctly passed strings and the size of the grammar (See equation 1). The genetic operators used were crossover, mutation and reproduction.

$$F(G) = \sum_{i=1}^{\#ofsentences} (LENGTH(S_i)/MAXPARSE(G, S_i)) - 1 \quad (1)$$

The training set included 20 simple English sentences consisting of noun phrase (NP) and a verb phrase (VP) structures.

Given the symbolic character of grammar induction, genetic programming is apt for this purpose. However, like in the study conducted by Javed [6] it was found that standard genetic programming was unable to evolve solution grammars. This is due to the high interdependency among the subparts of a context-free grammar. The grammar must be seen as a whole structure not as a structure made up of building blocks. Therefore the aim of the study was to formalise a control module for the genetic programming. A second shortcoming of the standard genetic programming technique is the tree representation of the individuals. The tree representation served to be an impediment in the search for the solution grammar. The reasoning behind this statement is that as mentioned before, the productions in a grammar are highly interdependent meaning that a slight change in one element can dramatically change the fitness of the whole grammar. The researchers stress that crossover and mutation could cause more harm than good in the evaluation of grammars. The situation mentioned in the research has been looked at by the pioneers of genetic programming and is termed destructive crossover [4]. The solution employed by Korkmaz and Üçoluk [11] uses the same reasoning as non-destructive crossover where an offspring created by crossover or mutation forms part of the next generation only if its fitness is better or equal to its parents. .

Rodrigues and Lopes [12] worked on the induction of context-free grammars. The grammars in the research were represented as tree structures with the root node being 'S' which represents the start symbol. The initial population was randomly generated. All the productions were reachable directly or indirectly from the start symbol 'S'. The fitness measure was computed by multiplying the *specifity* and *sensitivy*. Specifity is the total number of correctly rejected true negatives divided by the number of true negatives and false positives in the training set. Sensitivity is calculated by the total number of true positives correctly accepted by the grammar divided by the number of true positives and false negatives in the training set. The experiment in this study makes alterations to the standard genetic programming technique with the introduction of two new operators; the 'Incremental Learning Operator' as well as the 'Expansion Operator'. The Incremental Learning Operator was brought in to support convergence in the genetic run. The operator adds new positive productions through the use of the table produced when constructing the Cocke, Younger and Kasami CYK algorithm. The CYK algorithm is a parsing algorithm for context free grammars. It takes in a grammar in Chomsky Normal Form (CNF) as well the string to be passed. The string is constructed from the bottom up using the productions in the grammar [12]. For a more detailed explanation see [13]. Table I shows an example of the table produced by the CYK algorithm. Once the table is constructed, the Incremental Learning Operator checks to see if there are any productions in the top row of the table. If there are, a new production is created by changing the left

production of the production in the table to 'S' (start symbol of the grammar). If there are no productions in the top row then the new production is created by making the start variable the left hand side production and the right hand side variables are created by using the left hand side of the productions in the row below the top row. This operator adds positive productions which then promote convergence; however it does not guarantee the rejection of false negatives.

TABLE I. EXAMPLE OF CYK TABLE FOR THE STRING 'ABA'

2	S → C D		
1	C→ A B	D→B A	
0	A → a	B→ b	A→ a
	A	**b**	**a**

The Incremental Learning Operator was successful in adding positive productions to the grammar [12]. The expansion operator adds diversity to the population by creating new productions and adding them to the grammar. To mitigate against the creation of useless productions, the left hand side of the production rule is a new variable and the right hand side contains variables that are in the grammar as well as a new variable. Two new productions are created through the expansion operator. The expansion operator was then used to replace the reproduction operator of standard generational genetic program. The approach was successful in inducing grammars for the Tomita set of benchmark languages.

Based on the literature review, it was found that genetic programming is able to induce small structured grammars. Through the analysis of previous research, the importance of understanding and aligning the requirements of grammar induction with those of genetic programming come through. Domain appropriate representation of the grammar, genetic operations, parameters and evaluation function are important in the design of a successful genetic programming system.

The next section describes the genetic programming algorithm implemented in the study presented.

V. GENETIC PROGRAMMING ALGORITHM

The aim of the current study is to explore whether genetic programming can be used for evolving grammars. The evolved grammars will be used to generate password cracking guess words. Therefore the first step is to analyse the capability and performance of genetic programming for the induction of grammars. The current section will therefore be centered on the topic of genetic programming for grammar evolution. Previous research has investigated the application of genetic programming on a few grammar languages. In this section a genetic programming system will be presented where the genetic programming technique is applied to fifteen benchmark regular language sets and eleven context-free grammar languages.

A. Datasets

The datasets used were designed for the induction of deterministic finite automata (DFA). Since DFAs are used as acceptors for context-free languages, it is appropriate to use these datasets to induce grammars that generate the context-free languages.

The datasets for the first seven regular languages were created by Blair and Pollack [14]. The target DFA's are small in size, meaning the language can be represented using a DFA containing one to four states. The remaining seven regular languages were created by Miclet and Gentile [15]. The instances contained in the dataset include both positive (belong to the language) and negative (do not belong to the language) words.

TABLE II. REGULAR GRAMMAR LANGAUGES

	Language Description
L1	a *
L2	(a b) *
L3	Any sentence without an odd number of consecutive a's after an odd number of consecutive b's.
L4	Any sentence over the alphabet a, b without more than two consecutive a's.
L5	Any sentence with an even number of a's and an even number of b's.
L6	Any sentence such that the number of a's differs from the number of b's by 0 modulo 3.
L7	a * b * a * b *
L8	a * b
L9	(a * + c *) b
L10	(a a) * (b b b) *
L11	Any sentence with an even number of a's and an odd number of b's.
L12	a (a a) * b
L13	Any sentence over the alphabet a, b with an even number of a's.
L14	(a a) * b a *
L15	b c * b + a c * a

The dataset for context-free languages was created by the researcher by generating the positive examples and negative instances. These languages have also been used in previous studies [16].

TABLE III. CONTEXT-FREE GRAMMAR LANGUAGES

	Language Description
L1	$a^n b^n$ where n ≥0
L2	$a^n c b^n$ where n ≥0
L3	All palindromes with an odd number of letters. ∑ = {a, b}
L4	$s s^r$ where s is in (a + b) *
L5	$s c s^r$ where s is in (a + b) *
L6	s is in (a + b)*: the number of a's in s is equal to the number of b's in s.
L7	$a^n b^{2n}$ where n ≥0

L8	$\{a^n b^n: n \geq 0\} \cup \{a\}$
L9	a a * b a *
L10	$a^n b^{2n}$ where $n \geq 1$
L11	All strings with balanced brackets. $\sum = \{(,)\}$

A. *Representation*

The individuals for the current program are represented as tree structures with the maximum of arity of two as shown in Fig 2. A maximum width and depth allowed for the control in size of the individuals in the population to avoid bloat as suggested by Mernik et al. [9].

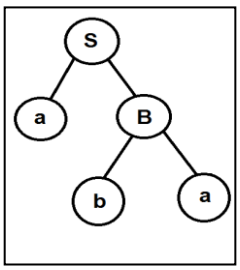

Fig. 2. Tree representation for the grammar productions: S→a B, B→b a

The root node in each tree is always 'S' which represents the start symbol. This allows for all the nodes to be reached directly or indirectly from the start symbol. The function set used were the variables 'S', 'A','B','C','D' and 'E'. The number of variables was sufficient because none of the target languages required more than five states. Having a small set of variables also reduced the search space. The terminals were mined from each sample set. The values include '0'and'1' where the $|\Sigma| = 2$. The terminal symbol '2' was added where the $|\Sigma| = 3$. As in the study conducted by Korlmaz and Göktürk [11], the productions of a grammar are highly interlinked; therefore to address the need for unity, the single tree representation permits the tree to evolve as a unit.

B. *Initial Population*

The initial population was randomly generated using the grow method. Each individual represented a single grammar and matched the structure in Fig 2.

C. *Fitness Evaluation*

To evaluate the population, the accuracy calculation, which is commonly used in machine learning, was used [12]. The first step is computing a confusion matrix. A confusion matrix is a matrix that keeps count of the positive examples that the grammar can generate (True positive), the negative examples that the grammar cannot generate (True Negatives), the positive examples that the grammar cannot generate (False Negatives) and the positive examples the grammar failed to generate (False negative). The dataset of positive and negative instances are used to calculate these values.

To compute the confusion matrix, each example string in the dataset had to be evaluated to see whether or not it can be constructed using the given grammar. The Cocke, Younger and Kasami (CYK) algorithm was used for this evaluation given its previous success when implemented by Rodrigues and Lopes [12]. Firstly each individual was converted to Chomsky Normal Form .

Accuracy was then computed according to the formulae:

$$\frac{(True\ positive + True\ Negatives)}{(True\ posivive + True\ negative + False\ positive + False\ negative)} \quad (2)$$

Accuracy is a good measure in this case because it assumes equal cost for false positives and false negatives. This is appropriate because in grammar induction, it is equally important that the grammar being induced can generate only words in the language but must not generate words that do not belong to the language. The fitness value for each individual is equal to the accuracy measure.

D. *Selection Method*

The tournament selection method as described by Koza [4] is implemented. Tournament selection randomly selects individuals equal to the tournament size. The individual with the best fitness value is chosen as the winner of the tournament selection**.**

E. *Genetic Operators*

Genetic Operators allow the winners of the selection method to undergo genetic modification.

Reproduction: The reproduction method is implemented as follows: A grammar is selected using the selection method. That grammar is copied over to the next generation.

Crossover: Standard crossover as described by Koza [4] was implemented. To mitigate against bloat, a restriction in the size of the number of the tree is set at 15 for regular languages and 20 for the context-free languages. This number is kept low because when the tree is converted to Chomsky Normal Form for evaluation, the number of productions increases.

Mutation: Standard mutation as described by Koza [4] was implemented. The same bloat mitigation strategy was also applied to crossover.

VI. Experiment Setup

The genetic programming algorithm was implemented in Java using JDK1.7.0_45 and the simulations were ran on an Intel Core i7-3770 machine running Windows 7, 64-bit. Generation genetic programming was implemented. The parameter values of the genetic programming runs were tuned using trial and error and were set as follows for the regular languages: mutation= 15%, reproduction=5% and crossover=80%. A population size of 150 was set for all languages and the number of generations was set to 50. Ten runs were performed for each language.

The parameter values for the context-free languages were as follows: mutation= 20%, reproduction=5% and crossover=75%. A population size of 250 was set for all

languages and the number of generations was set to 50 -100. Ten runs were performed for each language.

VII. Results

The presented system was able to evolve solutions for thirteen out of the fifteen regular languages. For the context-free languages, the GP algorithm managed to evolve grammars for ten out of the eleven languages. Table IV shows a summary of the results as well as the production rules evolved for each language.

TABLE IV. Grammar Solutions Evolved

	Regular Language Solutions
L1	S → 0; B → S; S → 00; E → S0; S → BE
L2	S → 01; A → 01; D → AS; S → D
L3	No solution found
L4	S → D; S → AA; S → C; S → 0;S → S; S → B; S → SB; A → S; A → A; A → 0; B → D; B → 1B; B → S; C → 1; C → 01; D → 1S; D → CS
L5	S → 11; D → 01; S → 00; D → 10;D → SD; S → DD; S → SS
L6	C → 00; C → 1; C → 1; C → 1; S → C0; S → 01; B → SS; A → B; S → A; C → CS; D → CC; S → CD
L7	D → 0; C → D; D → 0; D → 0D; D → D1; A → 1; S → A; D → S; S → DD; B → 0; B → SB; S → B; S → CS
L8	D → 1; C → D; D → 0C; S → D ;B → 0S; S → B
L9	B → 1; C → 2B; B → C; S → B; C → 2S; B → C; B → C; S → B; S → 0S; S → S; S → S; S → 0S;S → S
L10	C → 00; D → 11; S → D1; S → 00;B → S; S → SB; S → S; S → CS
L11	No solution found
L12	C → 00; D → C; B → 01; B → B; B → 0B; S → B; S → DS
L13	D → 1; S → D; A → 0; A → SA; S → 0A; S → 1;C → SS; C → C; S → 1;C → CS; C → C; S → C
L14	B → 00; A → 1; C→ BA; S → C0; S → S0; S → S; C→ S; A → C; C → A; S → C0; C → S; S → C; A → S;S → A
L15	A → 0; A → A2; C → 1; C → 2C; C → C; S → 1C; D → 12; A → SD; B → A; S → AB; S → S
	Context-Free Language Solutions
L1	S → C; C → 01; B → 0S; S → B1
L2	S → C1; E → 2; D → E; S → D; C → 0S
L3	No solution found
L4	B → 0 ;B → 1; S → 10; B → S1; S → BB;B → 0S; D → BB; S → D
L5	E → 02; D → E; S → 2; S → DS; C → 0; E → SC; E → E; S → 0E; C → 1S; S → C1
L6	S → 10; S → 01; B → S0; S → 1B; S → SS
L7	A → 11; C → A; S → 0C; C → 11;E → SC; S → 0E
L8	A → 0; C → 0; D → C1; C → AD; S → C1; S → S; A → S1; S → A
L9	E → 0; S → E1; S → S0; S → 0S;S → S
L10	A → 11; C → A; S → 0C; C → 11; E → SC; S → 0E
L11	E → 1; C → 1; B → 0C; S → EB; C → 1; B → 01; S → CB; C → S; C → SC; S → 0C

VIII. Analysis

This section analyses the performance of genetic programming for grammar induction. As can be seen from Table V the number of runs where a correct solution was found is show. This value is referred to as the success rate. The success rate of genetic programming was higher simple languages such as L1 in the regular languages and L1 of Context-free languages.

TABLE V. Success rate for grammar induction

	Number of runs where a correct solution was found	
	Regular Languages	**Context-free Languages**
L1	10	10
L2	10	10
L3	0	0
L4	4	4
L5	1	1
L6	10	6
L7	1	4
L8	10	5
L9	6	10
L10	10	9
L11	0	10
L12	10	
L13	10	
L14	10	
L15	1	

The common characteristic of the languages that were not generated is that they contain an odd number of alphabet characters. This could be a challenge for genetic programming. A possible reason for this is that generally, it takes more productions to represent an odd number of variables as compared to an even number.

The genetic programming algorithm performed well in situations where the language being generated was simple such as L1, for both regular and context-free languages. The evaluation of context-free grammars was a challenge because of the lack of a benchmark data set. Initially the use of randomly generated datasets yielded unsatisfactory, brittle solutions. In those situations more positive and negative examples were added in order to guide the genetic programming algorithm towards a better solution.

IX. Conclusion

As shown by the results, genetic programming was successful in inducing a solution grammar for twenty three out of the twenty six languages that it was tested on. Therefore genetic programming can be applied to grammar evolution.

The biggest limitation in evolving grammars is the dependency of an algorithm on the dataset used. This problem was also experienced by the current investigation. The grammars produced were over specific. Over specification to the dataset causes the grammars to allow words that are not in the language to be generated by the solution grammar. To solve this problem domain knowledge was used to add more positive and negative examples to the dataset.

X. Future Work

Future work will look to apply the genetic programming algorithm obtained from the current study to a dataset made up of leaked passwords in order to evolve grammars to be used to generate password guess words. Once the grammars have been constructed, they will be used to generate passwords structures also known as password masks.

References

[1] M. Bishop, and D. Klein, Improving system security via proactive password checking. Computers & Security, 1995. 14(3): p. 233-249.

[2] M. Jakobsson, and M. Dhiman, The benefits of understanding passwords, in Mobile Authentication. 2013, Springer. p. 5-24.

[3] M. Weir, A. Sudhir, M. Breno, and G. Bill, Password cracking using probabilistic context-free grammars. in Security and Privacy, 2009 30th IEEE Symposium on. 2009. IEEE.

[4] J.R. Koza, Genetic Programming: vol. 1, On the programming of computers by means of natural selection. Vol. 1. 1992: MIT press.

[5] J.R. Koza, Introduction to genetic programming tutorial: from the basics to human-competitive results. in Proceedings of the 12th annual conference companion on Genetic and evolutionary computation. 2010. ACM.

[6] F. Javed, Inferring context-free grammars for domain-specific languages. in Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2005. ACM.

[7] F. Javed, Techniques for context-free grammar induction and applications. 2007. The University of Alabama

[8] M. Mernik, et al., Learning context-free grammars using an evolutionary approach. Birmingham: University of Maribor and University of Alabama, 2003a.(Technical Report), 2003.

[9] F. Javed, B. Bryant, M. Črepinšek, M. Mernik and A. Sprague, Context-free grammar induction using genetic programming. in Proceedings of the 42nd annual Southeast regional conference. 2004. ACM.

[10] P. Wyard, Context free grammar induction using genetic algorithms. in Grammatical Inference: Theory, Applications and Alternatives, IEE Colloquium on. 1993. IET.

[11] E. Korkmaz, and G. Üçoluk, Genetic programming for grammar induction. 2001.

[12] E. Rodrigues, and H.S. Lopes. Genetic programming for induction of context-free grammars. in Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on. 2007. IEEE.

[13] D. Klein, and C.D. Manning. Natural language grammar induction using a constituent-context model. in NIPS. 2001.

[14] A.D. Blair, and J.B. Pollack, Analysis of dynamical recognizers. Neural Computation, 1997. 9(5): p. 1127-1142.

[15] P. Dupont, and L. Miclet, Inférence grammaticale régulière: fondements théoriques et principaux algorithmes. 1998.

[16] A. Naidoo, Evolving Automata Using Genetic Programming, Masters Thesis 2008, University of KwaZulu-Natal, 2008