Minimal Combination for Incremental Grammar Fragment Learning

Nurfadhlina Mohd Sharef Trevor Martin Yun Shen

Artificial Intelligence Group, University of Bristol
Bristol, BS8 1TR, United Kingdom
Email: ennms@bris.ac.uk, trevor.martin@bris.ac.uk, yun.shen@bris.ac.uk

Abstract-*The ability to identify text fragments can benefit many information extraction tasks. The main challenge in this matter is to recognise the multiple patterns that exist across possibly heterogeneous sources. Traditional machine learning methods can generate grammars for a set of training examples but the disadvantage is their potential heavy computational overhead. The proposed incremental grammar learning method offers immediate learning of new examples by slight alteration of the existing grammar collections by measuring the changes needed to equalize two grammars. This minimal combination approach avoids overgeneralization and maintains the coverage of newly learnt and past examples without a need to repeat the training process. Our experimental results suggest that the proposed algorithm outperforms a fuzzy grammar learning algorithm implemented in normal supervised learning setting.*

Keywords- evolving fuzzy grammar, grammar fragments, incremental learning, grammar similarity, tagging, text mining.

1 Introduction

The "digital obesity" phenomenon is a pressing problem at both the individual and corporate level [1]. This is partly because information is recorded using different formats for similar purposes and according to different hierarchies and categories. However, it is likely for the objects placed within similar categories to have similar properties and with some degree of correspondence.

These information sources often contain text segments with additional structure that can be utilized to assist in the identification of matching items. For example, within a text segment such as: 'Burger King, 12, Meredith Lane, Ipswich IP2 3AX Ipswich' lies an embedded structure: business name, number, street name, placename, post code and county name while a brief comment such as: 'the Voyager has stopped receiving signal' contains information on the product name and problem description.

Works on lexical semantics and word disambiguation have studied text structures by focusing on linguistic patterns using formal language models i.e in the subject-verb-object structure. However, these methods are normally well-suited for texts that are in complete grammatical form. On the other hand, text segments such as addresses, dates and times, names of products, people, as well as simple sentence forms such as questions, complaints, and news are frequently contained within "casual" text such as email, posts on a web forum, text messages, etc which often does not follow formal grammar rules.

Humans are able to use clues that identify structured segments without necessarily following a strict pattern. There can be many variations of the patterns and humans are normally better at seeing the answer in a specific case than being able to define a pattern that is universally applicable. As such, this is an ideal problem for machine learning; the key subproblem is converting small sections of texts into a more structured form by using grammar fragments to add tags. Note that we are not attempting to parse the entire piece of free text, merely to locate sections within it that can be tagged with an additional structure. For this reason, we refer to grammar fragments, as they are not intended to be a complete language model.

We define grammar fragment learning as the extraction of text fragments that comply with the predefined grammar. In contrast to in-depth whole sentence understanding, the grammar fragments learning focus only on portions of text that are relevant to specific domains. For example, UK address grammars might extract the following patterns:

a. Chicago Rock Cafe Northgate St Ipswich IP1 3BX Suffolk
b. Lattice Lodge Guest House 499 Woodbridge Rd Ipswich
c. 25 St Nicholas Street Ipswich Suffolk IP1 1TW

while grammar collections for posts on a telecom provider forum will recognize the following question entries:

a. Can I use an apple with BT Broadband Talk and Softphone?
b. When does BT plan to develop and support a Softphone client for NON windows platforms such as Mac OSX or Linux?
c. What is Softphone and how much does it cost?

Several authors have proposed the use of genetic algorithm [2,3,4,5], fuzzy [6,7,8], semantic net [9] and hybrid probabilistic [10] to learn grammars mostly from structured examples but these follow the traditional route of training a fixed system on a set of examples, followed by application of that system. Adaptation to new examples (when the training set is found to be inadequate) will be computationally expensive and hence inefficient. The training set needs to be sufficiently diverse to cover all unseen examples or the training phase will need to be repeated in order to improve the training set. Thus, it is highly desirable for a method that can support the incremental learning of a new example by slight modification of the learned rules without reconsidering past examples.

The purpose of this study is to investigate an evolutionary grammar fragment learning with *incremental* feature i.e. it can be updated each time a new pattern is encountered (or each time a string is read) without extensive re-computation as compared to other supervised learning methods. The evolved grammar ensures the ability to encompass the newly learnt, as well as the previously seen, examples. Main issues in evolutionary grammar learning are also highlighted which are defining the fuzzy grammar learning operators and its representation aspect. This algorithm will benefit

ISBN: 978-989-95079-6-8

works in intelligent text pattern recognition and its relevant applications such as intelligent agent and recommendation system.
The paper is organised as follows: the first part gives the introduction to incremental grammar learning while section two describes some of existing works relating to text pattern learning. The third section details the proposed evolutionary algorithm setting followed by observed results in section four. The final section summarises the investigated issues and some future works related to the study.

2 Related Works

Text annotation and text tagging are among the active text extraction research areas. Information tagging has been applied for sentence structure learning in question answering system [11] as well as specific domains including finding medical terms in legal text [12,13], detecting contact information and address [14] and gene and protein interaction information identification [15,16]. Among the popular techniques that have been employed are CFG, statistical, and evolutionary method.
Viola and Narasimhan [14] reported the advantage of marrying statistical natural language processing to the analysis of non-natural language text through the Discriminative Context Free Grammar (Discriminative CFG) approach which has extracted contact information more accurately than a similar conditional Markov model. The hierarchically structured CFG method [3,7,14] is more powerful than the formal English language model such as the subject-verb-object and noun-adverb-adjective model because these models have limitations when more complex sentence forms are involved.
The CFG method provides a rich collection of features that can measure and tag the properties of a sequence of tokens. Another advantage is that the CFG can propagate long range token dependencies efficiently. CFG method can also result in the detection of semantic information (encoded in the form of a parse tree) within a sentence - for example the word *fish and chips* in the restaurant address entry can be classified as broad as a type of business, or a food business genre, or even as specific as a food name.
Backus-Naur-Form (BNF) is a simple method for specifying CFG. It consists of a set of production rules with various operators such as "[ ]" which means "*this part is optional*", "|" for *or*, etc. A BNF grammar is usually more easily deciphered than a regular expression. A method for generating BNF grammars from data was described in [17]. As with regular expressions, a BNF grammar does not generally allow partial matching and there are no methods for finding the similarity between two BNF grammars.
An alternative approach to this is the fuzzy grammar similarity algorithm [2,3,9] which can measure the degree of similarity between grammars. The incorporated fuzzy approach supports the uncertainty and redundancy notion in deriving the grammar. The fuzzy grammar method was applied in evolutionary grammar learning using genetic algorithm.
However, in common with other supervised learning algorithms, the grammars generated using GA method [2,3,4,5,6,10], can only recognize data belonging to the trained pattern and needs to be rebuilt to include newly added patterns. Our preliminary study on GA [6] shows that GA is unfeasible because of the slow convergence towards creation of a good parsing grammar. The genetic operators and random concept in GA had also ignored the semantic properties (sequence of the tokens that bears meaning) in the dataset.
The minimal combination approach discussed in this paper is a revised version from the maximal combination approach [6] which avoids overgeneralization. On the contrary, [6] has generated a lot of unseen examples which has also led to low accuracy. The preliminary experiment on genetic algorithm in [6] has shown that genetic algorithm method is unsuitable for the grammar learning task because it cannot support grammar similarity and is also burden by the slow convergence. Work described in this paper concentrates on an alternative minimal combination approach and demonstrates that different permutations of the training set have no significant effect on incremental evolutionary grammar learning.

3 Brief Introduction to Grammar Fragment Learning

The fuzzy parsing concept allows for the learning of the pattern in the strings and can be seen as an extension of sequence matching, in which: i) we allow one sequence to be defined by a grammar, so that the matching process is potentially more complex and ii) we allow the standard edit operations (insert, delete, substitute) on the string in order to make it conform to the grammar.
It is adapted from the Cocke-Younger-Kasami (CYK) algorithm to allow for:

a. fuzzy grammar fragments that can contain fuzzy sets of terminal symbols;
b. calculation of the overlap between fuzzy grammar fragments;
c. partial parsing, measured by the proportion of a string that has to be altered by Levenshtein-style operations in order to make it conform to a fuzzy grammar fragment.

We identify a grammar fragment with the set of strings that are parsed by the grammar fragment. Because we allow partial parsing, this is a fuzzy set. The fuzzy membership, $\mu_G(Si)$ calculates the similarity between string and grammar, estimated by the minimum number of edit operations needed to convert a string into one which is parsed by the grammar. A string's membership decreases as it needs more changes to make it conform to the grammar.

$$\mu_G(Si) = 1 - \min\left(1, \frac{\min_{P \in Ext(G)} (totalCost(Si, P))}{length(Si)}\right)$$

We define a cost to be a five-tuple *(I D S Rs Rt)* where *I*, *D* and *S* are nonnegative real numbers representing, respectively, the approximate number of insertions, deletions, and substitutions needed to convert a string parsed by the source grammar into one that satisfies the target grammar. *Rs* and *Rt* represent sequences of grammar elements remaining (respectively) in the source and target after the match; at least one of *Rs* and *Rt* is null in every valid cost. Table 1 shows a simple example of grammar similarity comparison between a source and a target grammar in a UK postal address. In this case, *Cost(sg,tg)=(0 1 1 countyName Null)* to substitute *word* with *placename*, and leave *word* and countyName unmatched.

ISBN: 978-989-95079-6-8

Table 1: Example of Grammar Edit Distance Operation
(*I:Insert, D:Delete, S:Substitute, Rs: remainder of source grammar, Rt: remainder of target grammar)

Source grammar, sg	Number	Word	Word	Streetending	Placename	
Target grammar, tg	Number	Placename		Streetending	Placename	Countyname
Edit distance*	0	S=1	D=1	0	0	Rt=1

The *totalCost* function defines a total order on costs by $C1 \le C2$ iff $totalCost(C1) \le totalCost(C2)$.

$$\text{totalCost}((I, D, S, Rs, Rt)) = I + D + S + \text{minLength}(Rs) + \text{minLength}(Rt)$$

Addition of costs is order dependent and is defined by

$$(I1, D1, S1, Rs1, Rt1) + (I2, D2, S2, Rs2, Rt2) = (I3, D3, S1 + S2, Rs2, Rt2)$$

where

$$I3 = I1 + I2 + \text{minLength}(Rt1)$$

$$D3 = D1 + D2 + \text{minLength}(Rs1)$$

For two grammars, *GS* and *GT*, *Overlap(GS, GT)* is the approximate degree to which an arbitrary string parsed by the source grammar *GS* is also parsed by the target grammar *GT*. We define the degree of overlap of a source grammar GS with a target grammar GT as

$$\text{Overlap}(GS, GT) = 1 - \min(1, \text{totalCost}(C)/\text{minLength}(GS))$$

where $C = CostGG(GS,GT)$ is an estimate of the cost of changing a string parsed by the grammar *GS* into one parsed by the grammar *GT*.

The *CostGG* function is defined as follows, where S and T are arbitrary sequences of grammar elements, *s*, *t* are terminal symbols, and *TSi* and *TSj* are (fuzzy) sets of terminal symbols.

$$\text{CostGG}(\text{nu}\mathbf{l}, T) = (0, 0, 0, \text{null}, T)$$

$$\text{CostGG}(S, \text{null}) = (0, 0, 0, S, \text{null})$$

$$\text{CostGG}(s, t) = (0, 0, \delta(s, t), null, null)$$

where $\delta(s, t)$ measures the cost of the symbol *s* replacing symbol *t*. By default $\delta(s, t) = 1$, if $t \neq s$ and $\delta(s, t) = 0$, if $t = s$.

A given grammar has full overlap with itself or any equivalent grammar, where we define equivalence as the ability to parse exactly the same set of strings. Conversely, two grammars have zero overlap if there is no string that both grammars can parse. For example, if *G1* defines postal address and *G2* defines email address then they have zero overlap as there are no strings that both can parse.

The degree of overlap is not symmetric, as can be seen if we consider two grammar fragments *G3* and *G4*, where the set of strings parsed by *G3* is a strict subset of those parsed by *G4* . In this case,

$$overlap(G3, G4) = 1$$

but

$$overlap(G4, G3) < 1.$$

Clearly, the degree of overlap could be determined by repeatedly generating and checking candidate strings—such an approach would be very computationally demanding.

Instead, we propose a method [2,3,9] based on estimating the number of edit operations needed to change an arbitrary string parsed by the first (source) grammar sequence into a string that would be parsed by the second (target) grammar. This will result the coverage of strings belonging to both grammars and is implemented through the grammar combination technique.

The former *maximal combination* technique [9] allows combination of grammars without filtering their cost. E.g, for the set of strings $S = \begin{Bmatrix} (a-b-c-d-e-f), \\ (a-b-a-c-d-e-f), \\ (b-c-d), \\ (a-d-e-f) \end{Bmatrix}$ the learned grammar is $G' = \{[a]-[b]-[a]-[c]-d-[e]-[f]\}$. Although this method can parse all the strings belonging to S, but it also allows for the parsing of strings which does not exist in S. Therefore our current method focuses on minimal combination; combination of grammars that have a difference of 1 and this have managed to avoid overgeneralization.

3.1 Incremental Evolutionary Grammar Learning Algorithm

As the fundamental of the study is to look at the pattern similarities within the strings, a parser that can tokenize and extract relevant symbols from each input is essential. This grammar derivation process will re-label the symbols with significant grammar markup (Fig. 2 and Fig. 3).

The incremental learning has allowed for the update of the available grammars when new pattern is triggered without needed to repeat the whole learning process (Fig.1). The employed minimal combination is governed by the cost between the source and the grammar; combination will only be executed when the change is one. This process contributes to the evolution of the whole grammar collection and the persistence of the parsing coverage.

A chosen grammar from the existing collection will be altered according to the grammar combination rules. Table 2 shows guidelines of grammar combination operations; insert, merge and create. These operations are based on the cost of changing the source to the target and the other way

Table 2: Grammar Combination Operation

Source Grammar	Target Grammar	Cost (Source,Target)	Cost (Target,Source)	Combination Operation	Combined grammar
a-b-c	a-b	0 1 0 - -	1 0 0 - -	Insert	a-b-[c]
a-b	a-b-c	1 0 0 - -	0 1 0 - -	Insert	a-b-[c]
a-b-c	a-B-c	0 0 0 - -	0 0 1 - -	Merge	a-B-c
a-B-c	a-b-c	0 0 0 - -	0 0 1 - -	Merge	a-B-c, B>b
a-F-c	a-G-c	0 0 1 - -	0 0 1 - -	Create	a-X-c , X:=F||G, $F \neq G$

ISBN: 978-989-95079-6-8

around. The *insert* operation is executed when the change based in the cost is a deletion or an insertion (Fig. 5). The combined grammar is always the altered version (with optional tags) of the longer grammar. When cost needs a substitute action and when generalization can be performed, *merge* operation is executed while *create* is executed when generalization is not suitable (Fig. 4). Note that the grammar combination is sensitive to the matched position of the grammar pairs. The notation '||' is only used as a convenience of the paper and not practised in the system development.

```
Algo. 1: Grammar Evolution
Input: S={s0,s1,…,sn}
Output: GT
Procedure:
initialise GT to empty set
gs = grammar derived from s[0] (Algo.3)
add gs to GT
for each string, s [i:1,...,n] in S[n]
  gt=grammar in GT that parses s with min cost C (Algo. 2)
  if C > 1 then
    Add gs to GT
  else
    if C = 1 and cost(gt,gs)=0 then
      replace gt with combine(gt,gs) (Algo.5)
    else if C = 0 and cost(gt,gs)=1 then
      replace gt with combine(gt,gs) (Algo. 5)
    else if C = 1 and cost(gt,gs)=1 then
      replace gt with combine(gs,gt) (Algo. 4)
    end if
  end if
```

Figure 1: Grammar Evolution Algorithm

```
Algo. 2: minimum parsing cost, C
Input: string,s, grammar gt[0... n]
Output: minGrammar and C
Procedure:
Initialise C to cost(s, gt[0] )
Initialise minGrammar to gt[0]
for each  [i:1,...n]
  if( C > cost(s, gt[i]) then
      C = cost(s, gt[i])
     minGrammar = gt[i]
  end if
end for
```

Figure 2: Maximum Parsing Cost Algorithm

```
Algo. 3: Grammar Derivation
Input:string, s
Input: terminal symbols representing S,
S ∈ {word, placename, streetending, number,...}
Output: gs, the grammar derived for s
Procedure:
initialise gs to empty string
initialise maxMem and mem to zero
for each token in s, s[i], [i:0,...n]
maxMem= fuzzy membership score of s[i] with t[0]
  for each terminal symbols, t[j], [j:1,...m]
    mem= fuzzy membership score of s[i] with t[j]
    if mem>maxMem and t[j] is less general than t[j-1] then
      append the terminal token into gs
    end if
  end for
end for
```

Figure 3: Grammar Derivation Algorithm

```
Algo. 4: Combine For Substitute Operation
Input: gs,gt
Output: combinedGrammar, subGrammar
Procedure:
initialise combinedGrammar and subGrammar to empty
string
Initialize n=lengthOf(gs)=lengthOf(gt)
for i=[0,…n]
  if gs[i] is equal to gt[i] then   combinedGrammar[i] = gs[i]
  else
     create  subGrammar= gs[i], subGrammar=  gt[i]
     append subGrammar into combinedGrammar[i[
end for
```

Figure 4: Combine (Substitute) Algorithm

```
Algo. 5: Combine For Insertion or Deletion Operation
Input: gs,gt
Output: combinedGrammar, subGrammar
Procedure:
initialize combinedGrammar, subGrammar  to empty string
if gs.length>gt.length then loop=gs.length
else loop=gt.length
for [i:0…loop]
   if gs[i] is equal to gt[i] then
      append gs[i] to combinedGrammar
  else
      if gs.length>gt.length then
         append gs[i] with optional tag to combinedGrammar
      else loop=gt.length
         append gt[i] with optional tag to combinedGrammar
  end if
end for
```

Figure 5: Combine (Insertion or Deletion) Algorithm

The impact of the grammar learning when the training set is prepared in various orders is being observed. Fig. 6 shows an example of a collection of grammars while Table 3 shows the grammar generated when the grammars are learned in the different orders as follows:

Order1: G_2-G_8-G_3-G_5-G_4-G_7-G_6-G_1.
Order 2: G_6-G_3-G_2-G_8-G_4-G_7-G_5-G_1
Order 3: G_2-G_5-G_3-G_8-G_4-G_7-G_6-G_1

Our observation has shown that the final set of evolved grammars can parse the same set of strings, with the same accuracy. Section 5 gives some remarks on this issue. However, further work is ongoing to investigate this in more detail from a theoretical and practical perspective.

$$G = \{G_1, G_2, ..., G_8\}$$
$$g_n = \{a, b, c, d, e, f, j, k\}, a > k > j$$
$$G_1 = w - n - w - se - pn - pc - cN$$
$$G_2 = w - w - n - w - se - pn - pc - cN$$
$$G_3 = w - w - acc - w - se - w - pn - pc - cN$$
$$G_4 = w - bt - w - se - pn - pc - cN$$
$$G_5 = w - w - n - w - se - w - pn - pc - cN$$
$$G_6 = w - w - acc - w - se - pn - pc - cN$$
$$G_7 = w - acc - w - se - pn - pc - cN$$
$$G_8 = w - w - bt - w - se - pn - pc - cN$$

Figure 6: Examples of training grammars

ISBN: 978-989-95079-6-8

Table 3: Evolved Grammars in Various Training Data Orders

Order 1	Order 2	Order 3
w-w-(bt||n)-w-se-pn-pc-cN ($G_2 \cup G_8$)	w-w-acc-w-se-[w]-pn-pc-cN ($G_6 \cup G_3$)	w-w-n-w-se-[w]-pn-pc-cN ($G_2 \cup G_5$)
w-w-(acc||n)-w-se-w-pn-pc-cN ($G_3 \cup G_5$)	w-w-(n||bt)-w-se-pn-pc-cN ($G_2 \cup G_8$)	w-w-acc-w-se-w-pn-pc-cN (G_3)
w-(bt||acc)-w-se-pn-pc-cN ($G_4 \cup G_7$)	w-(bt||acc)-w-se-pn-pc-cN ($G_4 \cup G_7$)	w-w-bt-w-se-pn-pc-cN (G_8)
w-w-acc-w-se-pn-pc-cN (G_6)	w-w-n-w-se-w-pn-pc-cN (G_5)	w-(bt||acc)-w-se-pn-pc-cN ($G_4 \cup G_7$)
w-n-w-se-pn-pc-cN (G_1)	w-n-w-se-pn-pc-cN (G_1)	w-w-acc-w-se-pn-pc-cN (G_6)
		w-n-w-se-pn-pc-cN (G_1)

4 Results and Discussion

The underlying ideas of the experiment are to examine approximate parsing, fuzzy overlap of grammar fragments, and their use in the incremental evolution of fuzzy grammars. To demonstrate the benefit of incremental evolution an experiment was carried out to compare the parsing scores of the grammars generated through fuzzy grammar learner (FG) versus the proposed incremental evolutionary learning grammar (IEG).

The term incremental is coined so that the grammar will always be able to parse all supplied string patterns (identified by grammars) including the triggered new string pattern. This simulates interaction with a human expert who would be able to correct errors—although clearly this disadvantages the FG method, which is not designed to undergo multiple training phases. On the other hand, the FG method is performed according to the normal supervised learning setting; the performance of the training dataset is measured by its parsing on the test dataset. No additional learning is allowed even the parsing score is lower than 1. Table 4 shows the terminal sets used in this experiment.

Table 4: Terminal Sets

Word (w)	Any token composed entirely of alphabetical characters
Number (n)	Any token composed of digits
Streetending (se)	One of {road, rd, avenue, ave, street, st, lane, ln} or similar
Placename (pn)	A handcrafted set of UK placenames
CountyName (cN)	A handcrafted set of UK district names
BusinessType (bt)	One of {café, restaurant, bar} or similar
Accommodation (acc)	One of {hotel, lodge} or similar
Alphanumeric (aN)	A token composed of alphabetical characters and digits

Two sets of address dataset in UK format are prepared; restaurant.xml and yellowpages.xml. The restaurant dataset contains a small set of restaurant addresses in the UK while the yellowpages dataset contains domestic addresses. Each dataset is split into 25% for training and 75% for testing. The restaurant dataset contains 85 entries while the yellowpages dataset contains 396 entries. Both have no repeated entries but naturally consist of similar or repeated string patterns. This experiment was repeated using three different orders of the training and test dataset, to test the validity of the results and the sensitivity and reliability of the algorithm. The same result was produced in the different data permutations. Future work on this study will focus on the formal theoretical proving of order independent learning to support this hypothesis.

Table 5 shows the comparison between the mean of maximum parsing score of the grammars learned. The average of exact parsing is a measure of the average of the maximum parsing score of the grammars on each string in the test dataset. Score of 100% means that all strings have been recognized by the grammar.

The IEG method generates new rules immediately and shows good incremental learning behavior as the grammar fragments evolve; meaning that the grammar fragments may be altered according to the change required to transform the source and target grammars so that they will be equally parsed with increased coverage. Thus, the IEG method will always give an optimal parsing score of 1 for each string it has encountered, as shown in Table 5. For each string in the dataset, there is either a grammar that can parse it partially or accurately; and a grammar can thoroughly parse a minimum of 1 string. This indicates that for every string in the dataset, there is at least a grammar that can parse it accurately (100% parsing). As opposed to the IEG method, the average of exact parsing in FG is much lower. Observation in the restaurant dataset shows that the minimum parsing score of the entries in FG is 0.667 and 0.75 in the yellowpages dataset. Both share a maximum of score=1.

As discussed in section 2, supervised learning method is constrained by its training stage; the training dataset should contain all the patterns expected to be seen in the test dataset. As shown in Table 5 the performance of the grammars generated in FG is lower than of IEG due to its insufficient training. The FG score in yellowpages dataset is higher than the restaurant dataset because the yellowpages dataset is larger than the restaurant dataset so during training a better coverage has been provided.

Table 5: Average of Exact Parsing

Setting	Dataset	Average of Exact Parsing (%)
FG	Restaurant	89.5
IEG	Restaurant	100
FG	Yellowpages	97
IEG	Yellowpages	100

This shows that the evolutionary learning method has successfully provided immediate and gradual learning to the grammars even when the training is found to be insufficient. The set of evolved grammars have successfully maintained the parsing coverage of the existing and newly combined

ISBN: 978-989-95079-6-8

patterns. The goal has been achieved where the adaptations are performed in light of new examples while preserving its current knowledge.

5 Summary and Conclusion

Incremental evolutionary grammar learning (IEG) is a method that can cope with newly found text pattern. It is an advantage over the expensive retrain-retest setting in traditional machine learning approaches when examples in the training phase are found to be insufficient. Empirical result shows that the IEG method has 100% parsing of the dataset compared to 89.5% and 97% achieved by the fuzzy grammar method. The incremental feature in IEG has also surpassed the genetic algorithm method both in the processing time and accuracy.

The underlying structure (grammar) of the text is first detected and corresponding grammar that represents the text is derived. The novel fuzzy grammar similarity is utilized to find similarity of the grammars. The evolution concept featured in this algorithm is reflected by the enrichment of the learnt patterns; where its incremental learning characteristic gradually updates its knowledge on the various patterns. Each time a new pattern is encountered, the adaptation is executed by a minimal adaptation which provides a *fit* coverage corresponding to the newly found and the altered grammar as well as all the strings that they belong. Over time, the set of evolved grammars become more concise, wider coverage yet sustaining its persistence.

Further work is ongoing to study the theoretical proving of the order-dependency effect on the evolved grammars, investigating a method to create more compact grammars and to expand the application of fuzzy grammar fragments to softer structures (e.g., identification of questions, negative comments, positive reviews, etc.).

References

[1] Martin, T. P. "Fuzzy sets in the fight against digital obesity," *Fuzzy Sets Syst.*, vol. 156, pp. 411–417, 2005.

[2] Petasis., G., (2004) eg-GRIDS: Context Free Grammatical Inference from Positive Examples using Genetic Search, International Colloquium on Grammatical Inference 2004, *Lecture Notes in Artificial Intelligence* 3264, pp. 223-234.

[3] Sakibara, Y., and Marumatsu. H., (2000) Learning Context-free Grammars from Partially Structured Examples, International Colloquium on Grammatical Inference 2000, *Lecture Notes in Artificial Intelligence* 1891, pp. 229-240.

[4] Smith, T.,C., and Witten, H., (1996) Learning Language Using Genetic Algorithm, *Lecture Notes in Computer Science* 1040 pp. 132-1445.

[5] Unold, O., and Cielecki, L.,(2007) Learning Context-Free Grammars from Partially Structured Examples: Juxtaposition of GCS with TBL, *Seventh International Conference on Hybrid Intelligent Systems*, pp. 348 – 352

[6] Mohd .S., N., Martin, T., Shen, Y. (2008) Incremental Evolutionary Grammar Fragments, *Proceedings of the UK Workshop on Computational Intelligence (UKCI 2008)*, pp. 89-94.

[7] Martin, T. and Azvine, B. (2006) Evolution of Fuzzy Grammars to Aid Instance Matching,. IEEE *International Symposium on Evolving Fuzzy Systems,* pp. 163 – 168.

[8] Martin, T, Shen., Y., Azvine, B. (2008) Incremental Evolution of Fuzzy Grammar Fragments to Enhance Instance Matching and Text Mining, *IEEE Transactions On Fuzzy Systems,* Vol. 16, No. 6, December 2008, pp. 1425-1438.

[9] Yuhua L., David M.L., Zuhair A. B., James D. O., and Keeley Crockett (2006) Sentence Similarity Based on Semantic Nets and Corpus Statistics,. *IEEE Transaction on Knowledge and Data Engineering*, Vol 18, No. 8 pp.1138-1150.

[10] Bosman, P. A. N. (2004) Learning Probabilistic Tree Grammars for Genetic Programming, Parallel Problem Solving from Nature - PPSN VIII, *Lecture Notes in Computer Science* Vol. 3242 pp. 192-201.

[11] Ravichandran, D., and Hovy, E.(2002) Learning Surface Text Patterns for a Question Answering System, *Proceedings of the 4th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 41-47.

[12] Dozier, C., et. al (2007) Fast Tagging of Medical Terms in Legal Text, *Proceedings Of The 11th International Conference On Artificial Intelligence And Law*, pp. 253-260.

[13] Serban, R., et al (2007). Extraction and Use of Linguistic Patterns for Modelling Medical Guidelines, *Artificial Intelligence in Medicine (2007) 39*, pp. 137-149.

[14] Viola, P. and Narasimhan, M., (2005) Learning to Extract Information from Semi-structured Text using a Discriminative Context Free Grammar,. *8th Annual International ACM SIGIR Conference*.

[15] Joshua, M. T, and Gilder, M. R., (2003) Extraction Of Protein Interaction Information From Unstructured Text Using A Context-Free Grammar, *Bioinformatics 19(16) Oxford University Press.* Vol. 19. No. 16. pp. 2046-2053.

[16] Tanabe, L., and Wilbur, W. J., (2002). Tagging Gene and Protein Names in Biomedical Text, *Bioinformatics Vol. 18, no. 8 2002*, pp. 1124-1132

[17] Mernik, M., Gerlic, G. , Zumer, V. and Bryant, B. R. (2003) Can a Parser be Generated from Examples, *Proceedings of the 2003 ACM Symposium on Applied Computing,* pp. 1063 –1067

ISBN: 978-989-95079-6-8