Incremental Evolutionary Grammar Fragments

Nurfadhlina Mohd Sharef
Artificial Intelligence Group
University of Bristol
BS8 1TR UK
ennms@bris.ac.uk

Trevor Martin
Artificial Intelligence Group
University of Bristol
BS8 1TR UK
trevor.martin@bris.ac.uk

Yun Shen
Artificial Intelligence Group
University of Bristol
BS8 1TR UK
Yun.shen@bris.ac.uk

Abstract

Existing work in smart text mining has used fuzzy grammar fragments to tag short sequences of text without needing a full language model. Grammar fragments can be created with standard machine learning approaches, but these generally use a two stage (train / test) approach that requires a fully representative set of training examples. Such approaches cannot easily cope with the addition of a new training example. In this paper we show that standard genetic approaches do not work well for learning grammars from examples of UK addresses, and propose an evolving algorithm which can incrementally learn grammar fragments without having to retrain on previously processed examples. It mixes a fuzzy overlap method with evolving grammar approximation operators, avoiding the heavy computational overhead of traditional genetic programming.

1. Introduction

Sentences are usually formed according to grammar rules. Grammar is defined as the study of how words and their component parts combine to form sentences [10]. The word order governs the message that is to be delivered in the sentences. A parser is a programme that has the ability to recognize sentences in accordance with a given grammar.

There are existing applications with the ability to label text fragments which are usually designed using the traditional machine learning train/test process [1,5,7,8,9,12]. This means that these systems can only classify unseen examples (unlikely examples encountered during the training phase) but do not have the ability to learn further. In cases where the training set is later discovered to be inadequate, the new examples must be added into the system and it needs to be re-built. Thus, it is desirable for a parser that can learn and adapt to variations of grammar fragments but still maintains the parsing ability of previously seen examples. In order to come out with such a parser, it is useful to compare grammars (or grammar fragments).

This research focuses on the parsing of segments of texts such as addresses, dates and times, names of products, people, as well as simple sentence forms such as questions, complaints, and news. These are usually contained within "casual" text such as emails, posts on a web forum, text messages, and the likes which often do not follow formal grammar rules. We intend to avoid the need for a full language model because it is difficult to specify, complex to process, and subject to problem domains.

In the case of postal addresses there may be official specifications – in the UK, for example, a correct address [4] requires (in order):

number and *street name*,
[*locality*],
town,
postcode (on a separate line) and
[*country*]

This example would suit '29 Meredith Rd Ipswich'. In practice, many similar patterns are allowable such as '21 London Rd Ipswich Suffolk IP1 2EZ' or 'Belfairs Hotel 33 Graham Rd Ipswich'. The variations of the pattern will probably increase as more data samples are encountered. The elements with bracket tags are optional in the grammar, which is one of our methods to cover generalization and parsing ability of all the datasets.

This research proposes an incremental evolving algorithm that extends the capability of the normal machine learning method especially in learning and adapting to new examples but still maintaining parsing ability of past examples. The fuzzy grammar approach used aims to extract structured section from texts and represent them in fuzzy grammar fragments.

The first part of this paper gives an introduction to incremental evolving grammar. The second part discusses related work under this area followed by the implementation of the proposed algorithm. The final part concludes the current findings and discusses some potential future works.

Source string	W	E	D	N	E	S	D	A	Y
Target string	T	U			E	S	D	A	Y
Edit distance*	S=1	S=1	D=1	D=1	=	=	=	=	=

Table 1: Example of string edit distance operation (*I:Insert, D:Delete, S:Substitute)

Source grammar	Number	Word	Word	Streetending	Placename	
Target grammar	Number	Placename		Streetending	Placename	Countyname
Edit distance*	=	S=1	D=1	=	=	I=1

Table 2: Example of Grammar Edit Distance Operation (*I:Insert, D:Delete, S:Substitute)

2. Related Works

Grammar approximation is a branch under the grammatical inference discipline. It is closely related to sentence matching, grammar parsing and evolving algorithms. This section will briefly discuss each of the mentioned areas and conclude with summary of the highlights.

2.1. Levenshtein Edit Distance

The Levenshtein edit distance (also known as edit distance) is the finest studied case of error model [6]. It defines the smallest number of insertions, deletions, and substitutions (I,D,S) required to change one string or grammar into another. Edit distance is also known as the approximate string matching model and can measure similarity between strings and grammars. Examples of the application of this method are shown in Table 1 and 2. This algorithm is usually applied using dynamic programming. It recursively finds the smallest cost of changing the source string to target string among the cost on the left, top and diagonal cell in the table matching matrix.

2.2. Grammar Representation

Additional structures within free texts can be utilized to assist in identification of matching items. The grammar parser represents the data in context-free grammar form rather than using the normal approach in linguistic where a sentence is determined based on its subject-verb-object structure. This approach is also more appropriate for the UK address datasets being used which are shorter sentences and semi-structured in nature.

The context-free grammar method is used because it can encode human understanding of data. A grammar can consist of non-terminal and terminal tokens. The non-terminal tokens are initialized to help in tokenizing the sentences in order to build the grammar. These include definition of number, street ending, post code, and so on. These definitions act as guidance for the basic structure of our dataset entries and are represented in a partial order table.

2.3. Grammar Parsing

Parsing a sentence involves the use of linguistic knowledge of a language to discover the way in which a sentence is structured. Exactly how this linguistic knowledge is represented and can be used to understand sentences is one of the questions that has engaged the interest of psycholinguists, linguists, computational linguists, and computer scientists.

In contrast to sentence parsing, grammar parsing requires the checking (parsing) of a grammar according to the defined grammar/structure. Numerous methods have been introduced in the grammar parsing research including tagging-based information extraction [3,11], document distributions and statistical model [1], evolutionary genetic algorithms [1,7,8,9], semantic nets [12], and fuzzy methods [4,5] These methods however are aimed at generating grammars that would parse defined dataset and have not covered the extension properties of the grammars.

Crisp values of grammar parsing for a sequence of symbols are either *valid* or *invalid*. Fuzzy set is introduced to allow partial parsing of a sequence of symbols, which enables us to get *the degree of closeness* to a grammar. Fuzzy parsing [4] can be seen as an extension of sentence matching, in which one sentence is defined by a grammar and as a result the matching becomes more complex. [5] introduced a novel algorithm measure of overlap between fuzzy grammar fragments which can be used to compare the range of two fuzzy grammar fragments (i.e. to estimate and compare the sets of strings that fuzzily conform to each grammar) without explicitly parsing any strings. A set of grammar fragments can evolve incrementally while guaranteeing that it will still be able to parse previously seen examples.

2.4. Genetic Programming

Genetic Programming (GP) is one of the evolutionary algorithms and follows Darwin's Theory of evolution. It is often paraphrased as *survival of the fittest* with population of computer programs that reproduce with each other using standard genetic operators which include crossover

and mutation. Over time, the best individuals will survive and eventually evolve to do well in the given environment.

The application of GP method in grammatical inference is mostly towards generating rules or parser for a specific dataset [1,7,8,9]. However, this method is not well suited for this research. Using the common elitist concept, some of the good candidates are eliminated especially during the early phase owing to the random concept widely used in this approach. The GP method is also time consuming and does not provide incremental learning support.

2.5. Summary

Numerous approaches in grammatical inference have successfully extended the practicability of sentence matching applications in text-related research. A great deal of contributions have been demonstrated especially in improving the semantic notion of text similarity [2,4,11]. However, except for the work proposed by [4], none of the discussed works have touched on the incremental evolution of grammar. The advantage of this idea is a more *intelligent* grammar in the sense of its ability to parse previous and new sentences. Further work on this idea is to be investigated in this research.

3. Evolutionary Grammar Learning

Since the basic underlying method of this idea is an evolving approach to parsing the grammar, testing practicability of genetic algorithm in the problem is fairly considerable. The setting of this experiment is inspired by [9] but in this experiment a binary tree is used instead. The evolutionary population prepares the environment in searching of the approximate grammar.

The initial population is generated which consists of 10 groups of files; where each group has a minimum of 1 and a maximum of 10 files. This initial population is then tested using membership function. The maximum score of each string is collected and the fitness of a file is the average maximum membership score of the strings. The fitness of a group is the summation of all files in a group. Elitist approach is used for mating the grammars within and across the files and also cross-groups. Standard genetic operations are then performed on them.

This procedure continues until all groups have been converged i.e membership score of the files in the groups have been increased and uniformed; grammar element components become more meaningful and can parse the dataset with higher membership score.

Figure 1 shows the generated grammar files scores in generation 0 while Figure 2 depicts the grammar files scores in generation 20. Note the wide range of the group membership score (fitness function) in generation 0 whereas in generation 20 the scores of the groups begin to converge with more groups having higher fitness function indicating the grammar elements can parse the dataset better. Figure 3 shows the grammar files scores in generation 32 where at this stage all the grammar files have become matured and stopped evolving at the maximum fitness of 0.4 and minimum of 0.2.

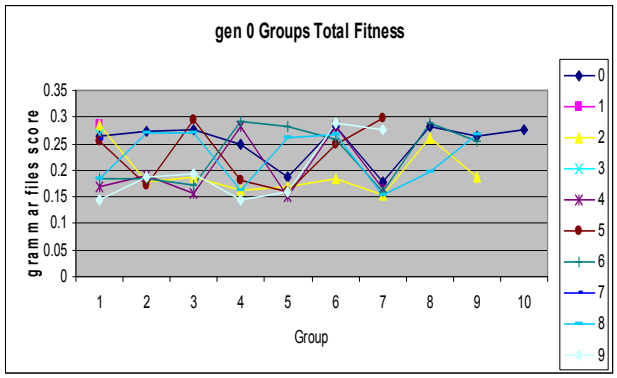

Figure 1: Grammar files scores in generation 0

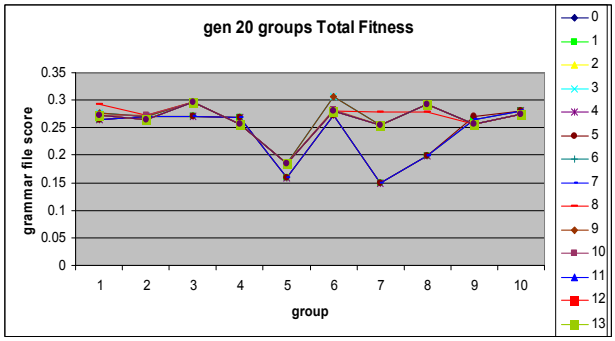

Figure 2: Grammar files scores in generation 20

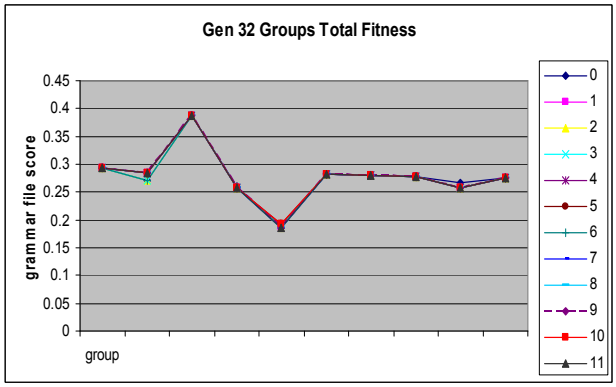

Figure 3: Grammar files score in generation 32

A few main weaknesses are detected from this experiment:

a. The genetic programming approach is not suitable to perform the incremental learning task.
b. It takes too long for the grammar files to converge, and the fitness is still

considerably low, as the best fitness is around 0.6.

Therefore, effort is taken to find ways to replace the common concept in existing evolutionary approach especially its random feature and the genetic operators.

4. Incremental Evolutionary Grammar Learning

The main goal of the experiment is to investigate ways to extract grammar fragments and incrementally learn the grammar sets so that it will be able to parse new as well as past examples. This algorithm aims to escape from the use of random concept and the use of common genetic operators. Instead, incremental grammar fragment learning operators are introduced. Another important step taken is the use of the fuzzy overlap function as a guide in order to generalize the grammars. Preliminary experiments show that this method is capable of creating grammars that correctly identify all addresses in the example set and requires less execution time than the genetic algorithm approach.

4.1 Fuzzy Partial Grammar Parsing

A disadvantage of template and normal grammar-based methods is that they do not cope well with close matches which are strings that almost conform to the grammar. Such strings should have partial membership in the extension of the grammar. Partial parsing is allowed in the grammar identification because we are dealing with data that is not always well-formatted and may be noisy. The UK address partial order table is as in Figure 4.

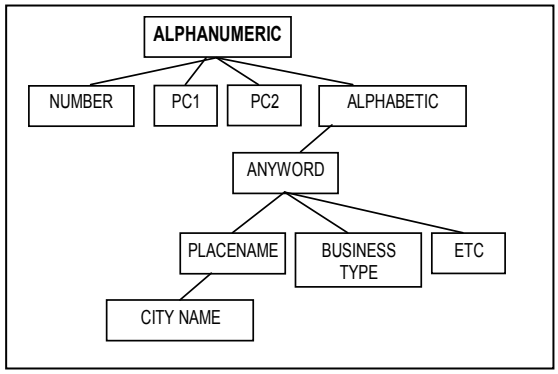

Figure 4: UK Address Partial Order Table

A partial parse [4] occurs when a sequence of tokens "nearly" conforms to a grammar so that the string can be parsed after small changes are made. The partial order table is one of the closest methods to human natural justification in choosing a more generalize conceptual element. It shows most of the general symbol at the top, with subclass below. This method is quite similar to the taxonomy approach where it is defined as the classification of organisms in an ordered system that indicates natural relationships [10].

4.2 Fuzzy Overlap Grammar

The fuzzy membership calculates the similarity between string and grammar. The pair has lower similarity if the string needs more changes to make it conform to the grammar. Estimation of similarity between grammars fragments are done by the fuzzy overlap function.

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*. 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 membership score is estimated by the minimum number of edit operations needed to convert a string into one which is parsed by the grammar.

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

where *totalCost(Si, P)* is the total cost of edit operations needed to convert a string *Si* into another string *P* that conforms to the grammar.

4.3 Incremental Grammar Learning

The overall algorithm for the generalization algorithm is as follows:

a. For each positive example, the grammar that parses the example with maximum membership is created. A new string will be converted to a grammar, G(S) and approximated grammar is tagged as G(T).
b. If the score of Overlap(GS,GT) is bigger than Overlap(GT,GS), that means G(S) is more general than G(T).
c. If the score of Overlap(GS,GT) is 1, that means G(S) can parse dataset belongs to G(T). If Overlap(GS,GT) is not equal to Overlap(GT,GS) then iteratively approximate the grammars as follows:
   i. If the grammar fragment consists of multiple contiguous optional elements, remove the optional tag and create a new optional fuzzy subset

$$[H_{new}]=\{g_i, g_{i+1}, \ldots, g_n\}$$

	Target grammar	Column=0	Column=1	Column=2	Column=3	Column=4	Column=5
Source grammar		**Null**	**number**	**anyWord**	**streetend**	**placename**	**postcode**
Row=0	**Null**	0 0 0	1 0 0	2 0 0	3 0 0	4 0 0	5 0 0
Row=1	**number**	0 1 0	**0 0 0 (E)**	1 0 0	2 0 0	3 0 0	5 0 0
Row=2	**placename**	0 2 0	0 1 0	**0 0 1 (D)**	1 0 1	2 0 1	4 0 1
Row=3	**streetend**	0 3 0	0 2 0	0 1 1	**0 0 1 (C)**	**1 0 1 (B)**	**3 0 1 (A)**

Figure 5: Overlap Matrix

ii. If the grammar fragment consists of multiple but non-contiguous optional elements, break the fragment into sub-fragments to maintain single optional rule and create fuzzy subset grammar accordingly with updated tag

$$H_{final}=[H_i]GH_{i+1}$$

iii. If the source grammar element is smaller or equal to the target grammar element according to the partial order table, it is replaced with the target grammar element. Otherwise, a new fuzzy grammar fragment with same head tag is created consisting of both the source and target element. The fragment is merged into the existing fragment to construct the final approximated grammar. The minimal generalization prevents for overgeneralization in the approximated grammar

$$Hi:=\{g_i, g_{i+1},\ldots, g_n\},\ g_i = \text{moreGeneral}_{(g_S,g_T)}$$

iv. If any disjoint fragments in the grammar can be tagged by existing rules then replace the fragment with relevant tag.

4.3.1 Matrix Crawling

Figure 5 shows an example of overlap matrix crawling. The elements on the row are the target while the elements on the columns are the source. The cell at row=1 and column=1 with overlap cost (0 0 0) shows a perfect match between the source and target, as this cell refers to grammar element 'number' at both the source and target element. The overlap calculation is repeated to find the minimum of the overlap cost for each element on the source and target until the final cell with row=3 and column=5. These dynamic programming-based functions automatically determine the minimum cost of changes needed to make the aligned grammars equal.

The changes needed to approximate the source and target (approximated grammar) can be identified by traversing the overlap matrix scores. The matrix traversing process starts from the final cell at row=3 and column=5, marked (A) in Figure 5 because in this cell the whole of the source and target grammar fragments have been calculated. Each minimum cell is marked with a bold uppercase alphabet. These cells are determined based on selecting the smallest cost of cells on the intermediate left, top and diagonal position of the current cell.

A grammar element is made optional (given square bracket tags) when the next minimum cost is on the intermediate left cell or the cell above current cell while the cell on the diagonal position means element on the source and target are substitute pair. The optional element means that the element is not compulsory in the grammar. The substitute element indicates the element can be swapped with its pair to allow the grammar to parse the entry.

The matrix is crawled starting from the final cell (3,5)**.** Its neighbouring cells have the cost (1 0 1), (2 0 1) and (4 0 1). The smallest cost between these three are the left cost at row=3 and column=4 with value (1 0 1). This process iterates until row=0 and column=0 is reached. Once the matrix traversing is completed, the source and target can be approximated. The following is the approximation based on grammars in Figure 2:

X:=placeName
X:=anyWord
G1:= placeName-postCode
Addr:= number-X-streetend-[G1]

The compound grammar G1 stores the contiguous optional element to maintain only 1 single optional element in each approximated grammar. The elements tagged with brackets are optional. X indicates a substitute pair where in this case placeName in the source should be substituted by anyWord to make the grammars parsing equal.

The partial order table in Figure 4 is used to find more general element in substitute pair. Elements at the upper branch are more general than the lower. In this example, anyWord is more general than placename. The substitute pair is only generalized if the target element is more general than the source element. Therefore, the final approximation becomes:

G1:= placeName-postCode
Addr: Number-anyWord-streetend-[G1]

Figure 6 shows more examples on address approximation. Note as new address is encountered, the final address changes the source grammar so that it would be able to parse strings belonging to the target grammar.

Address	Grammar derived from address	Approximated Grammar
107 hatfield rd ipswich ip3 9ag	number-placeName-streetend-placeName-postCode	ADDR:=number-placeName-streetend-placeName-postCode
121 sidegate ln ipswich	number-anyWord-streetend-placeName	G1:=streetend-placeName-[postCode] G2:=anyWord G2:=placeName ADDR:=number-G2-G1
alnesbourne priory club nacton rd ipswich	anyWord-anyWord-anyWord-anyWord-streetend-placeName	G1: =streetend-placeName-[postCode] G2:=anyWord G2:=placeName G3: =anyWord-anyWord-anyWord G3: =number ADDR:=G3-G2-G1

Figure 6: Example of grammar approximation

5. Conclusion and Future Works

The incremental ability is the main highlight of this research and the proposed algorithm has proven ability to understand new grammar structures and sustain the precision of the grammar representations. The fuzzy overlap method adopted has successfully estimated the similarity between grammars as to measure the need for generalizing the grammar. The approximation operators had escaped from the common genetic operators and as a result managed to accomplish the approximation task correctly for all the past grammars as well as new data. Future works will touch on the practicability of the algorithm on more free structure texts as longer texts may have richer properties. Another open problem would also be to investigate method to expand or enrich the partial order table.

6 References

[1]. 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.

[2]. Erhard R., and Philip A. Bernstein (2001) A survey of approaches to automatic schema matching,. *The VLDB Journal 10: 334-350 (2001) / Digital Object Identifier (DOI) Springer Verlag.*

[3]. 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 2003. pp. 2046-2053.

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

[5]. Martin, T., (2005) Fuzzy Sets in the Fight Against Digital Obesity, *Fuzzy Sets and Systems, Vol* 156 pp. 411-417.

[6]. Navarro, G., (2001) A Guided Tour to Approximate String Matching, *ACM Computing Surveys*.

[7]. 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.

[8]. 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.

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

[10]. The American Heritage, Dictionary of the English Language, Fourth Edition Published by Houghton Mifflin Company. All rights reserved. © 2006 by Houghton Mifflin Company.

[11]. 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*.

[12]. Yuhua Li. David Mc. Lean, Zuhair A. Bandar, James D. O'Shea, 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.