International Journal of Computer Applications (0975 – 8887)
Volume 23– No.4, June 2011

Parsing with Neural and Finite Automata Networks: A Graph Grammar Approach

Sanjay Bhargava
Department of Computer Science
Banasthali University
C-62, Sarojini Marg, C-Scheme, Jaipur - 302001

G. N. Purohit
Dean, AIM & ACT
Banasthali University
Banasthali, Rajasthan - 304022

ABSTRACT

Parsing with finite automata networks implies, in one way, the conversion of a regular expression into a minimal deterministic finite automaton, while parsing with neural networks involves parsing of a natural language sentence. In ‘Parsing with finite automata networks’ finite automata are frequently combined using a set of rules for various operations like *union, concatenation, and kleene closure*; while in ‘Parsing with neural networks’ an incremental tree is usually obtained, by using a set of rules for connecting a possible parse tree to the previously obtained incremental tree. Apparently, all the above rules that are being applied in parsing *whether with finite automata networks* or *with neural networks* belong to some graph transformation rules. These rules depict a new concerned area of grammars known as graph grammar, that is, a grammar that operates on graphs. This research paper presents a twofold investigation on the use of graph grammar as it explores an attempt to use both aspects of graph grammars (*to generate a valid language and to parse a language for its validity*) for parsing with (i) neural networks and (ii) finite automata networks.

General Terms

Neural networks, Finite automata networks, Backtracking, Incremental Parsing.

Keywords

Parsing, Graph grammar, Regular expression, Natural language processing.

1. INTRODUCTION & BACKGROUND TOPICS

Practically, all the approaches for parsing with natural languages use some type of neural network architecture and some typical statistical function towards getting a parse decision. However, the so obtained parse decision might be incorrect because of the unbounded and ambiguous nature of natural language grammars. So, keeping in view that no natural language parser is 100% correct (with respect to parse decisions), an attempt was made by Bhargava and Purohit [5] to reduce the parsing time, by parsing a natural language sentence without the use of any statistical function ***as an application of parsing with neural networks***.

Regular expressions and finite automata are two dissimilar representations for regular languages: Regular expressions (a finite or infinite set of strings of alphabet characters), on one hand, generate regular languages while, on the other hand, finite automata (graphs) accept regular languages. Apparently, regular expressions and all variants of finite automata (NFA with or without ε-transitions, or DFA) are equivalent because all of them represent the same language, that is, a regular language. Thereby, all of them are convertible into each other (see, e.g. [28]). It is a well-established fact that each regular expression can be transformed into a nondeterministic finite automaton (NFA) with or without ε-transitions (see, e.g. [51], [57], and [59]). In addition, there also exist algorithmic approaches to convert a regular expression into DFA by use of intermediate NFAs (see, e.g. [11] and [28]). In the recent past, efforts have been made towards conversion of a finite set of strings into a minimal, deterministic, acyclic finite-state automaton (see, e.g. [9] and [15]). Unfortunately, all these attempts either convert a finite set of strings into a DFA or they convert a regular expression into a minimal DFA using the following chain of conversion: *regular expression* $\rightarrow$ *NFA with ε-transitions* $\rightarrow$ *NFA without ε-transitions* $\rightarrow$ *DFA* $\rightarrow$ *minimal DFA*. Hence, it has been found that there is no standard method available to convert a regular expression into a minimal DFA directly, that is, without the use of any intermediate NFA. To overcome this problem, an attempt was also made by Bhargava and Purohit [4] to convert a regular expression into a minimal DFA directly without the use of any intermediate NFA ***as an application of parsing with finite automata networks***.

Graph grammars play an extensive and vital role in the above parsing algorithms (see, e.g. [4] and [5]). In the present research paper we investigate the wide-ranging role of graph grammars in parsing with neural and finite automata networks. In order to make a fair introduction to this theme, we describe in brief some technical processes and other related topics.

1.1 Finite Automata Network

An *automata network* is a collection of automata connected together according to a directed graph ([42]). The vertices of this directed graph are considered as automata and the edges indicate the existence of communication links. This graph has no parallel edges. Each automaton can change its state at discrete time steps as a local transition function of the states and a global input, and synchronous action of the local state transition defines a global transition on the entire network.

A *finite automata network* is a type of “automata networks” which consists of finite automaton as vertices (nodes). A finite automaton is a mathematical model of a system, with discrete inputs and outputs. The system can be in any one of a finite number of internal configurations or states. The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs. Thereby, a finite automaton is a model of behavior composed of a finite number of states, transitions between those states, and actions. A state stores information about the past, that is, it reflects the input changes from the start of system to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment.

1.2 Neural Network

A *neural network* is a system of programs and data structures that approximates the operation of the human brain. A neural network usually involves a large number of processors operating in parallel, each with its own small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially "trained" or fed large amounts of data and rules about data relationships. A program can then tell the network how to behave in response to an external stimulus or can initiate activity on its own (within the limits of its access to the external world). Thereby, a neural network is a massively parallel distributed processor made up of simple processing units which has a natural propensity for storing experiential knowledge and making it available for use. A neural network resembles the brain in two respects, that is, first by "acquiring knowledge from its environment through a learning process", and then "storing the acquired knowledge by using interneuron connection strengths, known as synaptic weights". Hence, a neural network is a type of artificial intelligence that attempts to imitate, in a way, how a human brain works. Instead of using a digital model in which all computations manipulate "zeros and ones", a neural network works by creating connections between *processing elements*, the computer equivalent of neurons.

Therefore, a neural network is made of individual units termed neurons ([48]). Each neuron has a weight associated with each input. A function of the weights and inputs (typically, a squashing function applied to the sum of the weight-input products) is then generated as an output. These individual units are connected together as shown in Figure 1, with an input layer, an output layer and usually one or more hidden layers.

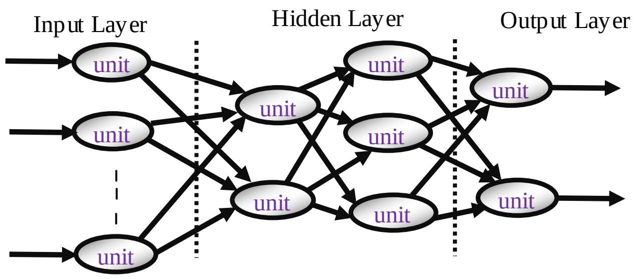

Fig 1: A typical feedforward neural network.

Typically, the input layer consists of one unit per attribute and the output layer of one unit per class. The number of units and topology within the hidden layer is obtained by the respective grammar. Through algorithms such as backpropagation, the weights of the neural net can be adjusted so as to produce an output on the appropriate unit when a particular pattern at the input is observed. The backpropagation algorithm works by running the training instance through the neural network, and calculating the difference between the desired and actual outputs. These differences are then backpropagated from the output layer to the hidden and input layers in the form of modifications to the weights of each of the component neurons. This modification is done in a manner proportional to the contribution to the difference in output, so that the weights most responsible for the difference are modified the most.

Feedforward neural networks cannot take labeled trees as input because they neither can deal with structured objects that have variable size nor they can embed relations among atomic constituents. In principle, *recurrent neural networks* might be employed by converting the tree into a sequence (see, e.g. [25] and [33]).

1.2.1 Recurrent Neural Network

A *recurrent neural network* (RNN) is a modification to feedforward neural network architecture to allow temporal classification, as shown in Figure 2. In this case a "context" layer is added to the structure which retains information between observations. At each time-step, new inputs are fed into the RNN. The previous contents of the hidden layer are passed into the context layer. These then feed back into the hidden layer in the next time-step. So, during parsing, RNN is the best suitable architecture to store the intermediate incremental trees information for getting the final parse tree for a sentence.

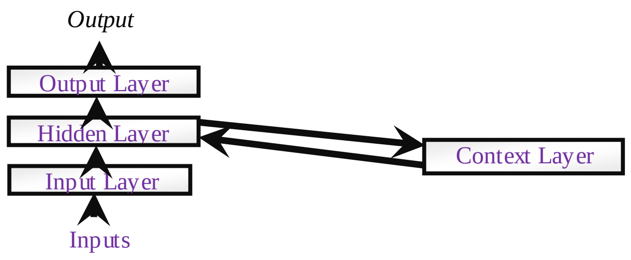

Fig 2: A Recurrent neural network architecture.

1.3 Parsing

According to Aho *et al.* [2], *parsing* is the process of determining whether a string of tokens can be generated by a grammar. To analyze a sentence or a language statement, parsing breaks down words into functional units that can be converted into machine language. Parsing involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output. The grammatical phrases of the source program are represented by parse tree. A *parse tree* is a tree that represents the syntactic structure of a string according to some formal grammar. In a parse tree the interior nodes are labeled by nonterminals of the grammar while the leaf nodes are labeled by terminals of the grammar. Parse trees may be generated for sentences in natural languages as well as during processing of computer languages such as programming languages. A *parser*, capable of constructing parse tree, is a program, usually part of a compiler that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts that can then be managed by other programming. A parser may also check that all necessary input has been provided.

1.4 Graph Grammars

A *grammar* is a set of rules which can

- o generate a construct from a list of terminals;

- recognize that a construct obeys the grammar rules.

In natural languages, a construct is a sentence and the terminals are words. The grammar can specify such notions as subject-verb agreement, the declaration of nouns and the conjugation of verbs and the syntactic structure of sentences. In computational linguistics, our area of interest is to find a natural language grammar so that machine could recognize and interpret the sentences. In natural language parsing, sentences are treated as trees (graphs) and the parsing algorithm tries to obtain incremental trees (graphs) using a set of graph grammar rules. At each step of parsing, *graph grammar* maintains the legally correct connection between two trees to get an enhanced one. Thereby, all the connection process (of two trees) takes place under the supervision of *graph grammar rules* defined for this purpose.

According to McCreary [44] and Rozenberg [50], a *graph grammar* is a grammar that operates on a graph. In a graph grammar, graphs are generated from some initial graph by replacing a sub graph of the host graph by another graph, and embedding the inserted graph into the host graph. Thereby, a graph grammar is a canonical generalization of a context free string grammar. Its core is a finite set of productions of the form (A, R, C), where A is the label of the replaced vertex, R is the graph from the right hand side, and the connection relation C specifies the embedding of the right hand side R into the local environment of the replaced vertex. Graph grammars were invented, in early seventies, in order to generalize (Chomsky) string grammars. Graph grammars were originated from Chomsky grammars by substituting the replacement of strings with the replacement of graphs. The main idea was that of extending concatenation of strings to a “gluing” of graphs. The action of gluing two graphs is a construction, in the category of *graphs* and *graph morphisms*, called *pushout*. A graph, defined as an algebraic structure, is a 6-tuple (V, E, s, t, l_V, l_E), where

- V and E are two finite sets,
- $V \cap E = \varnothing$,
- s, t : $E \rightarrow V$ are two mappings indicating the source and the target of an edge,
- l_V: $V \rightarrow \Sigma_V$, and l_E: $E \rightarrow \Sigma_E$ are two mappings from V and E in two finite sets of labels.

Given two graphs G = (V, E, s, t, l_V, l_E) and G′ = (V′, E′, s′, t′, $l_{V'}$, $l_{E'}$), a graph morphism μ from G to G′ is a pair (μ_1, μ_2), μ_1: $V \rightarrow V'$, μ_2: $E \rightarrow E'$ such that μ satisfies the following two conditions: (i) labels are preserved i.e. $\mathbf{l_V(v_i) = l_{V'}(\mu_1(v_i))}$, and (ii) incidence is preserved i.e. $\mathbf{\mu_1(s(e_i)) = s'(\mu_2(e_i))}$.
It is obvious that there is a morphism from a graph H to a graph G if G contains H. For the graphs G and G′, as shown in Figure 3, there is a graph morphism from G to G′ as G′ contains G; while for the next pair of graphs G and G′, as shown in Figure 4, there is no graph morphism because neither of the graphs contains the other.

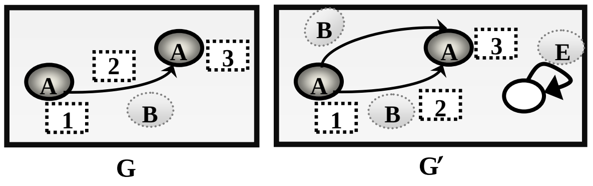

Fig 3: G′ contains G, showing that there is a morphism from G to G′.

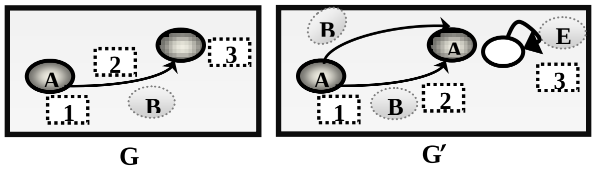

Fig 4: Neither of the graphs contains the other, showing that there is no morphism.

1.5 Incremental Parsing

Incremental processing of natural language is based on the intuitive fact that natural languages are processed from left to right, and most of the times are processed incrementally (see, e.g. [21] and [38]). The *incrementality hypothesis* implies that the semantic interpretation of some initial part of the sentence, to be parsed, is available as the initial scan of the input material proceeding from left to right. The next step towards incremental parsing is to further add a semantic interpretation into initial semantic interpretation with the aim of getting a better semantic interpretation for a larger part of input sentence. And this process continues till we get an interpretation for the whole sentence.

For incremental parsing, parse trees need to be connected incrementally. The interaction rules during connection between parse trees are described in various studies (see, e.g. [40], [46], [54], [55], and [56]). The interaction between the parse trees occurs at the word level and forces the syntactic analyzer to keep an entirely connected structure at all times. Parsing proceeds from left to right through a series of incremental trees, each spanning one additional word to the right. Let us have a sentence s = $w_1w_2w_3...w_{|s|}$ and a parse tree T for it. All the pendant vertices of T are labeled by words and the remaining vertices (of degree 2 or more) are labeled by nonterminal symbols. The recursive definition of T_i (for i $\in$ {1, 2, ..., |s|}) spanning $w_1w_2...w_i$ is as follows:

- T_1 consists of the chain of vertices and edges from w_1 to its maximal projection.
- T_{i+1} consists of all the vertices and edges in T_i and the chain of vertices and edges from w_{i+1} to L, where L is either a vertex of T_i or the lowest vertex of T dominating both the root of T_i and w_{i+1}.

Moreover, we also have the following definitions towards finding the incremental trees for s:

- The connection path for w_{i+1} is the difference between T_{i+1} and T_i. In other words the connection path for w_{i+1} is the parse tree that has to be attached to T_i to get T_{i+1}.
- A vertex that is both in T_{i+1} and T_i is called the host.
- The vertex labeled by the POS tag of w_{i+1} is called a foot.

To accommodate the next input word w_{i+1} to the incremental tree T_i, a connection path is computed. There can be more than one such connection path. A selection procedure selects the best connection path to obtain the next incremental tree T_{i+1}. The better is the selection procedure, the less time it will take to compute T_{i+1}. A selection procedure chooses the best connection path and host to generate the next incremental tree T_{i+1}. Once we obtain T_{i+1}, we look ahead to get T_{i+2}, based on T_{i+1} and next word w_{i+2}, in exactly the same way as we obtained T_{i+1} and this process continues till either we get T for s or we fail to get it.

Figure 5 shows the sequence of incremental trees for a sentence of the corpus.

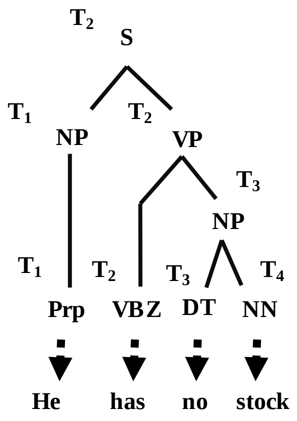

Fig 5: The incremental trees of the sentence "He has no stock". Vertices are labeled with the incremental tree that includes them for the first time.

1.6 Backtracking

With respect to the concept of trees, *backtracking* picks the first path and tries to go into its depth, searching for the solution; if it fails, then it steps backwards and picks the next path (another alternative), and so on. As a result, we can conclude that in the end the outcome is a resulting vector that contains the solutions. Backtracking solves problems that can be narrowed down to finding some sort of Descartes multiplication of N elements in series, which can be combined to satisfy some strict inner constraints between each other. It's our job to analyze how we can "re-phrase" the problem to some sort of series. It means that we must "code" the solution into a vector. It is also our job to identify the source elements of series from which we're going to build the possible paths of that so-called tree, or just a simple vector where we are moving forward trying to combine the elements from the source array, and on each failure taking one step backwards and trying the next alternative. We do this until a valid solution is found or we fail to get it.

According to Kruse [34], backtracking attempts to complete a search for a solution to a problem by constructing partial solutions and always ensures that the partial solutions remain consistent with the requirements of the problem. Backtracking then attempts to extend a partial solution towards completion but when an inconsistency with the requirements of the problem occurs the actual backtracking starts by removing the most recently constructed part of the solution and tries another possibility. Figure 6 shows the process of backtracking by a depth-first search where there is just one root node.

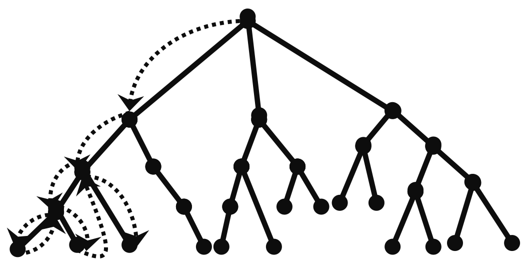

Fig 6: Backtracking: Depth-first search for a solution.

The contents of this paper are arranged as follows.
Section 2 deals with review of literature on related work and its current status. Next section 3 first details the significance of graph grammars in parsing with neural networks followed by a description on the role of graph grammars in parsing with finite automata networks. Last Section 4 deals with the conclusions of the present research paper.

2. RELATED WORK

Related work done in various areas by learned scholars is divided into five sections: graph grammars and its applications; conversion of regular expression into finite automata; parsing with neural networks and natural language processing; incremental parsing; and neural networks and recurrent neural networks.

2.1 Graph Grammars and its Applications

As far as we know, very few algorithms exist for the inference of graph grammars and its applications in parsing. Johnson *et al.* [30] described a variant of the subset construction that required the unique determination of the states in which a semantic action was required. The use of graph grammars for syntactic pattern recognition by providing an enumerative method for inferring a limited class of context-sensitive graph grammars was described by Bartsch-Spörl [3].

Briscoe *et al.* [8] described a substantial grammar to produce a general purpose morphological and syntactic analyzer for English language. Details of graph grammars and its various applications were enumerated by Ehrig [18] and Rozenberg [50]. Furthermore, McCreary [44] stated that the objects created through productions of a graph grammar were graphs rather than strings, showing that the theory of graph grammars is a generalization of formal languages and the theory of string grammars. All the above references were helpful for understanding the graph grammar concepts in the present study.

Abney [1] first defined stochastic attribute-value grammars and then gave an algorithm for computing the maximum-likelihood estimate of their parameters. Palm [47] described the role of graphs in natural language parsing. Later, Cook and Holder [14] presented a scheme for discovering frequent substructures in graphs. Schmid [52] gave a model on parsing using graph grammars, which was a generative probability model for unification-based grammars in which rule probabilities depended on the feature structure of the expanded constituent. Finally, Kukluk *et al.* [35] observed that graph grammars combine relational aspects of graphs with the iterative and recursive aspects of regular expressions.

2.2 Conversion of Regular Expression into Finite Automata

There are very few methods available for conversion of a regular expression into a minimal DFA without use of any NFA (see, e.g. [4]). After scanning the available related literature, all the reviewed methods fall into one of the following three categories.

(i) Conversion of a regular expression into an NFA (with or without ε-transitions).
(ii) Conversion of a regular expression into a DFA by using intermediate NFA.
(iii) Conversion of a set of strings (finite or infinite) into a finite automaton (NFA or DFA).

Besides the above three categories, Bhargava and Purohit [4] gave a method to convert a regular expression into a minimal DFA without the use of any NFA. This method was an ideal example of parsing with finite automata networks using graph grammars.

2.3 Parsing with Neural Networks and Natural Language Processing

Tomita [58], Chapman [12], and Aho *et al.* [2] enumerated various parsing techniques, starting from the very fundamental topics of parsing up to its applications. A context-free parsing algorithm that can parse sentences with unknown parts of unknown length was designed by Lang [37]. Lavie [39] described a method on developing an integrated heuristic scheme for selecting the parse that was deemed "best" from such a collection. Later, another method to find the most probable parse tree from a valid set of such trees was formulated by Bod [6]. Charniak [13] and Manning and Schu̇tze [43], on syntactic parsing, enumerated statistical language processing techniques such as *word tagging, parsing with probabilistic context free grammars, grammar induction, and syntactic disambiguation*. A neural network based statistical parser which was trained and tested on the Penn Treebank was presented by Henderson [26]. In this parsing scheme, the neural network was used to estimate the parameters of a generative model of left-corner parsing, and these parameters were used to search for the most probable parse.

Bhargava and Purohit [5] presented a distinctive approach of parsing with neural network. In their approach, the use of statistical function was minimized and hence the parsing time was considerably reduced.

2.4 Incremental Parsing

Ghezzi and Mandrioli [21] and Larchevêque [38] enumerated the concept of incremental parsing which was followed and is modified in the present study. Regarding incremental parsing concept, Frazier [20], Yamashita [61], and Kamide and Mitchell [31] suggested that the human parser assembles substructures eagerly before reaching the structure head. Steedman [55], Stabler [54], Milward [46], Sturt and Crocker [56], and Lombardo *et al.* [40] had discussed the interaction rules during incremental connection between parse trees.
Lane and Henderson [36] described the use of Simple Synchrony Networks (SSNs) for learning to parse *English* language *sentences* drawn from a corpus of naturally occurring text, by means of an incremental parsing. An algorithm that simulated the building of structure by marking, during the scan of the tree, the branches which would be built by a perfectly informed incremental parser, was designed by Lombardo and Sturt [41]. Several issues related to incremental (left-to-right) beam-search parsing of natural language using generative or discriminative models, either individually or in combination, were enumerated by Roark [49].

2.5 Neural Networks and Recurrent Neural Networks

Neural Networks: Kitano [32] and Boers & Kuiper [7] described their approaches of grammar-based encodings of neural networks. The above approaches were helpful in "joining of graph grammars and neural networks" and are modified in the present study. Later, neural network parsers were enumerated by Miikkulainen [45] and Lane and Henderson [36]. Hebbian parsers ([23]) and holistic parsers ([27]) were two classical examples of neural network parsers. Patterson [48] defined a neural network as a composition of individual units called neurons.

Recurrent Neural Networks: A detailed overview on the use and features of various recurrent neural network architecture (RNN) was enumerated by Jain and Waibel [29], Elman [19], Sharkey [53], Wermter and Weber [60], Hammer and Tiňo [24], erňanský *et al.* [17], Grüning [22], Cartling [10], and Dobnikar and Šter [16]. Kolen and Kremer [33] used the concept of recurrent neural networks as advancement over feed forward neural networks. In a closely related description, Hawkins and Boden [25] stated that "as compared to feedforward neural networks recurrent neural networks generally performed better on sequence analysis tasks".

3. PARSING USING GRAPH GRAMMAR

In *Parsing with neural networks using graph grammars*, Bhargava and Purohit [5] provides an algorithm in which a sentence of natural language is taken as input and the algorithm decides whether it is syntactically correct or not. Bhargava and Purohit [5] made use of recurrent neural networks and incremental parsing in this method. At each step of incremental parsing, the correct connection path linking the next word to the incrementally build parse tree was predicted using a recurrent neural network model. A set of rules for all the graph operations required to obtain incremental trees for this parser graph grammar was devised and kept together in the form of an algorithm.

In *Parsing with finite automata networks using graph grammars*, Bhargava and Purohit [4] converts an input regular expression into a minimal deterministic finite automaton (DFA) without the use of any NFA by means of a parsing algorithm. A set of graph grammar rules, that is, the rules for union, concatenation, kleene closure, and minimization operations over DFA was designed for the above conversion and kept together in the form of an algorithm. During the conversion process graph operations took place using an appropriate graph grammar rule. This method exhibited an example of parsing with finite automata networks in a *dual manner*, that is,

(i) If the input regular expression was not valid then the algorithm would stop without producing the output DFA. Hence the input regular expression was parsed for its validity, which is the first aspect of parsing using graph grammars. Here the graph grammar belongs to *parser grammars* because it parses the regular expression.
(ii) If the input regular expression was valid then the algorithm would convert it into an equivalent minimal DFA which is in some way the second aspect of parsing using graph grammars. Here the graph grammar belongs to *generator grammars* because it constructs the DFA for the language extracted from regular expression thereby representing the language by a new structure.
The present research paper discusses in detail the implication of graph grammars with respect to parsing with the two networks, *viz.* neural and finite automata networks. For this we take two previous research works attempted by Bhargava and Purohit (see, e.g. [4] and [5]) and bequeath the authority of graph grammars during parsing.

3.1 Parsing with Neural Networks

From the basic results of natural language processing, parsing of natural languages, graph grammars and neural network architectures, Bhargava and Purohit [5] have derived a simple, efficient and innovative method for parsing with neural networks using graph grammars. They have presented a novel methodology for parsing a natural language, based upon the *incrementality hypothesis*, a generally held assumption about human parser. In addition, the method ([5]) also relies upon one more unique hypothesis, that is, "a sentence's parse tree has the best possibility to be accepted as a connection path in future if it is used as a connection path for most of the times in past".

Though in most previous attempts, a statistical function for obtaining a parse decision has been a necessary component; however, Bhargava and Purohit [5] removes the necessity of any such statistical function thus reducing the amount of parsing time. In addition, it also has been shown that the parsing time by the proposed parser ([5]) was shorter as compared to the parsing time of four most time-efficient EFD parsers hence, showing the supremacy of the proposed parser ([5]) over EFD parsers.

Bhargava and Purohit [5] dealt with a combination of two different but related streams, that is, *graph grammars* and *parsing of natural languages*, even though an almost equal attention was paid towards both the above streams in their method. For this, they emphasized on the use of graph grammars for parsing a natural language by applying it at each stage of parsing. Therefore, the method ([5]) showed its supremacy over the previous works of similar approach by paying concentration on two streams, that is, graph grammars and parsing of natural languages, unlike the other attempts which were aimed at only one stream that is, parsing of natural languages.

3.2 Parsing with Finite Automata Networks

From the basic results of formal language parsing, graph grammars and automata theory, Bhargava and Purohit [4] have derived a simple novel method to construct a minimal deterministic finite automaton from a regular expression. This method ([4]) removes the dependency over the necessity of lengthy chain of conversion, that is, regular expression $\rightarrow$ NFA with ε-transitions $\rightarrow$ NFA without ε-transitions $\rightarrow$ DFA $\rightarrow$ minimal DFA. Therefore the main advantages of the minimal DFA construction algorithm by Bhargava and Purohit [4] are its minimal intermediate memory requirements and hence, it's reduced time complexity. This algorithm ([4]) converts a regular expression of size n in to its minimal equivalent DFA in $O(n.\log_2 n)$ time. In addition to this, the time complexity is further shortened to $O(n.\log_e n)$ for $n \geq 75$.

Again, Bhargava and Purohit [4] dealt with a combination of two different but related streams, that is, *graph grammars* and *parsing with finite automata networks*, even though an almost equal attention was paid towards both the above streams in their method. For this, they emphasized on the use of graph grammars in the form of graph algorithms for conversion of regular expression into DFA. Therefore, the method ([4]) again had shown its supremacy over the earlier efforts of the same kind by paying concentration on two streams, that is, graph grammars and parsing of finite automata networks, unlike the other attempts which were aimed at only one stream that is, parsing of finite automata networks.

4. CONCLUSION & FUTURE WORK

In this study we use graph grammar learning for parsing with (i) finite automata networks and (ii) neural networks, hoping that the expressive power of graphs and the ability of graph grammars to generalize the string grammars will turn out to be an influential learning paradigm. Thereby, this twofold study will be able to discuss the inference of graph grammars in parsing with neural and finite automata networks.

The method by Bhargava and Purohit [5] dealt with a combination of two different but related streams, that is, graph grammars and parsing of natural languages, even though an almost equal attention was paid towards both the above streams in the method. The future applications of the above parsing of natural language may lead to the development of an efficient incremental parser which is informed by the network architecture in taking decisions about attachment ambiguity. Contribution of some other knowledge sources like semantic knowledge-base and a set of subcategorization frames, so that ambiguity is reduced, will also help in developing a parser for natural language.

Further, most researches attempted hitherto are based on the use of intermediate NFA for the conversion of regular expression into DFA. Further most of the researches have not focused on the role of graph grammar during such conversion. Bhargava and Purohit [4] presented the algorithm that uses intermediate DFA in place of NFA. The algorithm ([4]) was fully equipped with graph transformation rules and hence was emphasizing the graph grammar role in parsing with finite automata networks. The algorithm ([4]) had shown a time complexity which was shorter as compared to other available methods, thus motivating the use of DFA in place of NFA for similar studies.

Finally as the present research paper depicts a speckled role of graph grammars in parsing with neural and finite automata networks, its future applications are immediate with respect to some other automata networks like cellular automata networks. In future, we are planning to investigate the role of graph grammars in parsing with cellular automata networks.

5. REFERENCES

[1] Abney, S. P. [1997]. "Stochastic attribute-value grammars". *Computational Linguistics*. vol. 23, no. 4, pp. 597-618.

[2] Aho, A. V., R. Sethi, and J. D. Ullman [2001]. *Compilers: Principles, Techniques and Tools*. Pearson Education Asia. New Delhi.

[3] Bartsch-Spörl, B. [1983]. "Grammatical inference of graph grammars for syntactic pattern recognition". Lecture Notes in Computer Science no. 153. Springer-Verlag, Berlin/Heidelberg, New York. pp. 1-7.

[4] Bhargava, S. and G. N. Purohit [2011]. "Construction of a minimal deterministic finite automaton from a regular expression". *International Journal of Computer Applications (IJCA)*. vol. 15, no. 4, pp. 16-27.

[5] Bhargava, S. and G. N. Purohit [2011]. "Parsing a Natural Language: A Non-Statistical Approach". *National Journal of Computer Science & Technology (NJCST)*. vol. 3, no. 1, pp. 23-33.

[6] Bod, R. [1995]. “The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars”. In *Proceedings of the 7th Conference on European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Morgan Kaufmann Publishers, San Francisco, CA. pp. 104-111.

[7] Boers, E. J. W. and H. Kuiper [1992]. “Biological metaphors and the design of modular artificial neural networks”. Master’s Thesis. Leiden University, The Netherlands.

[8] Briscoe, E., C. Grover, B. Boguraev, and J. Carroll [1987]. "A formalism and environment for the development of a large grammar of English”. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI’87), Milan, Italy*. Morgan Kaufmann Publishers, San Francisco, CA, USA. pp. 703-708.

[9] Carrasco, R. C. and M. L. Forcada [2001]. “Incremental construction and maintenance of minimal finite-state automata”. *Computational Linguistics*. vol. 28, no. 2, pp. 207-216.

[10] Cartling, B. [2008]. “On the implicit acquisition of a context-free grammar by a simple recurrent neural network”. *Neurocomputing*. vol. 71, no. 7-9, pp. 1527-1537.

[11] Chang, C. H. and R. Paige [1992]. “From regular expressions to DFAs using compressed NFAs”. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*. Lecture notes in Computer Science no. 644. Springer-Verlag, London. pp. 90-110.

[12] Chapman, N. P. [1987]. *LR Parsing Theory and Practice*. Cambridge University Press. New York.

[13] Charniak, E. [1996]. *Statistical Language Learning*. Mass. [u.a.] : MIT Press. Cambridge, MA.

[14] Cook, D. J., and L. B. Holder [2000]. “Graph-based data mining”. *IEEE Intelligent Systems*. vol. 15, no. 2, pp. 32-41.

[15] Daciuk, J., S. Mihov, B. W. Watson, and R. E. Watson [2000]. “Incremental construction of minimal acyclic finite-state automata”. *Computational Linguistics*. vol. 26, no. 1, pp. 3-16.

[16] Dobnikar, A. and B. Šter [2009]. “Structural properties of recurrent neural networks”. *Neural Processing Letters*. vol. 29, no. 2, pp. 75-88.

[17] erňanský, M., M. Makula, and u. Beňušková [2007]. “Organization of the state space of a simple recurrent network before and after training on recursive linguistic structures”. *Neural Networks*. vol. 20, no. 2, pp. 236-244.

[18] Ehrig, H. [1988]. *Graph-Grammars and their Application to Computer Science*. Springer-Verlag, New York, Inc. Secaucus, NJ, USA.

[19] Elman, J. L. [1991]. “Distributed representations, simple recurrent networks, and grammatical structure”. *Machine Learning*. vol. 7, no. 2-3, pp. 195-224.

[20] Frazier, L. [1987]. “Syntactic processing: Evidence from Dutch”. *Natural Language and Linguistic Theory*. vol. 5, no. 4, pp. 519-559.

[21] Ghezzi, C. and D. Mandrioli [1979]. “Incremental parsing”. *ACM Transactions on Programming Languages and Systems*. vol. 1, no. 1, pp. 58-70.

[22] Grüning, A. [2007]. “Elman backpropagation as reinforcement for simple recurrent networks”. *Neural Computation*. vol. 19, no. 11, pp. 3108-3131.

[23] Hadley, R. F. and M. B. Hayward [1997]. “Strong semantic systematicity from hebbian connectionist learning”. *Minds and Machines*. vol. 7, no. 1, pp. 1-37.

[24] Hammer, B. and P. Tiňo [2003]. “Recurrent neural networks with small weights implement definite memory machines”. *Neural Computation*. vol. 15, no. 8, pp. 1897-1929.

[25] Hawkins, J. and M. Boden [2005]. “The applicability of recurrent neural networks for biological sequence analysis”. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*. vol. 2, no. 3, pp. 243-253.

[26] Henderson, J. [2003]. “Neural network probability estimation for broad coverage parsing”. In *Proceedings of the 10th Conference on European Chapter of the Association for Computational Linguistics - Volume 1*. Association for Computational Linguistics, Morristown, NJ. pp. 131-138.

[27] Ho, E. K. S. and L. W. Chan [1999]. “How to design a connectionist holistic parser”. *Neural Computation*. vol. 11, no. 8, pp. 1995-2016.

[28] Hopcroft, J. E. and J. Ullman [1979]. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Longman Publishing Company, Inc. Boston, MA, USA.

[29] Jain, A. N. and A. H. Waibel [1990]. “Incremental parsing by modular recurrent connectionist networks”. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann Publishers, San Marco, CA. pp. 364-371.

[30] Johnson, W. L., J. H. Porter, S. I. Ackley, and D. T. Ross [1968]. “Automatic generation of efficient lexical processors using finite state techniques”. *Communications of the ACM*. vol. 11, no. 12, pp. 805-813.

[31] Kamide, Y. and D. C. Mitchell [1999]. “Incremental pre-head attachment in Japanese parsing”. *Language and Cognitive Processes*. vol. 14, no. 5-6, pp. 631-662.

[32] Kitano, H. [1990]. “Designing neural networks using genetic algorithms with graph generation systems”. *Complex Systems*. vol. 4, no. 4, pp. 461-476.

[33] Kolen, J. and S. Kremer [2001]. *A Field Guide to Dynamical Recurrent Networks*. IEEE Press. New York.

[34] Kruse R. L. [1991]. *Data Structures and Program Design*. 2nd edn. Prentice Hall of India Private Limited. New Delhi.

[35] Kukluk, J. P., L. B. Holder, and D. J. Cook [2007]. “Inference of node replacement graph grammars”. *Intelligent Data Analysis*. vol. 11, no. 4, pp. 377-400.

[36] Lane, P. C. R. and J. B. Henderson [2001]. "Incremental syntactic parsing of natural language corpora with simple synchrony networks". *IEEE Transactions on Knowledge and Data Engineering*. vol. 13, no. 2, pp. 219-231.

[37] Lang, B. [1988]. "Parsing incomplete sentences". In *Proceedings of the 12th International Conference on Computational Linguistics Volume - 1*. Association for Computational Linguistics, Morristown, NJ. pp. 365-371.

[38] Larchevêque, J. [1995]. "Optimal incremental parsing". *ACM Transactions on Programming Languages and Systems (TOPLAS)*. vol. 17, no. 1, pp. 1-15.

[39] Lavie, A. [1994]. "An integrated heuristic scheme for partial parse evaluation". In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, NJ. pp. 316-318.

[40] Lombardo, V., L. Lesmo, L. Ferraris, and C. Seidenari [1998]. "Incremental processing and lexicalized grammars". In *Proceedings of 20th Annual Conference of the Cognitive Science Society, Madison, WI - USA*. Lawrence Erlbaum Associates, Publishers Mahwah, New Jersey, London. pp. 621-626.

[41] Lombardo, V. and P. Sturt [2002]. "Incrementality and lexicalism: A treebank study". In *Lexical Representations in Sentence Processing*. John Benjamins: Computational Psycholinuguistics Series - 2002. pp.137-156.

[42] Luerssen M. H. [2005]. "Graph grammar encoding and evolution of automata networks". In *Proceedings of the 28th Australasian Conference on Computer Science - Volume 38 (ACSC '05)*. Australian Computer Society, Inc. Darlinghurst, Australia. pp. 229-238.

[43] Manning, C. D. and H. Schütze [1999]. *Foundations of Statistical Natural Language Processing*. Cambridge, Mass. [u.a.] : MIT Press.

[44] McCreary, C. L. [1988]. "Parsing a graph grammar". In *Proceedings of the 1988 ACM 16th Annual Conference on Computer Science (CSC '88)*. ACM, New York, NY. pp. 249-255.

[45] Miikkulainen, R. [1996]. "Subsymbolic case-role analysis of sentences with embedded clauses". *Cognitive Science*. vol. 20, no. 1, pp. 47-73.

[46] Milward, D. [1995]. "Incremental interpretation of categorial grammar". In *Proceedings of the 7th Conference on European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Morgan Kaufmann Publishers, San Francisco, CA. pp. 119-126.

[47] Palm, A. [1999]. "The expressivity of tree languages for syntactic structures". *The Mathematics of Syntactic Structure: Trees and Their Logics*. The theory of syntactic domains, Technical Report no. 75, Department of Philosophy, University of Utrecht. pp. 113-152.

[48] Patterson, D. W. [2007]. *Introduction to Artificial Intelligence and Expert Systems*. Pearson Education in South Asia. New Delhi.

[49] Roark, B. [2004]. "Efficient incremental beam-search parsing with generative and discriminative models: keynote talk". In *Proceedings of the Workshop on incremental Parsing: Bringing Engineering and Cognition Together*. ACL Workshops. Association for Computational Linguistics, Morristown, NJ. pp. 16-17.

[50] Rozenberg, G. [1991]. Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific.

[51] Rytter, W. [1989]. "A note on optimal parallel transformations of regular expressions to nondeterministic finite automata". *Information Processing Letters*. vol. 31, no. 2, pp. 103-109.

[52] Schmid, H. [2002]. "A generative probability model for unification-based grammars". In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*. Association for Computational Linguistics, Morristown, NJ. pp. 1-7.

[53] Sharkey, N. [1992]. *Connectionist Natural Language Processing*. Intellect. Oxford, England.

[54] Stabler, E. P. [1994]. "The finite connectivity of linguistic structure". In *Perspectives on Sentence Processing*. Lawrence Erlbaum Associates, Hillsdale, New Jersey Hove, UK. pp. 303-336.

[55] Steedman, M. J. [1989]. "Grammar, interpretation and processing from the lexicon". In *Lexical Representation and Process*. MIT Press, Cambridge, MA. pp. 463-504.

[56] Sturt, P. and M. Crocker [1996]. "Monotonic syntactic processing: A cross linguistic study of attachment and reanalysis". *Language and Cognitive Processes*. vol. 11, no. 5, pp. 449-494.

[57] Thompson, K. [1968]. "Regular expression search algorithms". *Communications of the ACM*. vol. 11, no. 6, pp. 419-422.

[58] Tomita, M. [1985]. Efficient Parsing for Natural Language: a Fast Algorithm for Practical Systems. Kluwer Academic Publishers.

[59] Watson, B. [1995]. "Taxonomies and toolkits of regular language algorithms". Ph.D. Thesis. Eindhoven University of Technology, CIP-DATA Koninklijke Bibliotheek, Den Haag.

[60] Wermter, S. and V. Weber [1997]. "SCREEN: Learning a flat syntactic and semantic spoken language analysis using artificial neural networks". *Journal of Artificial Intelligence Research*. vol. 6, no. 1, pp. 35-85.

[61] Yamashita, K. [1994]. "Processing of Japanese and Korean". Ph.D. Thesis. Ohio State University, Columbus, Ohio.