Bio-Inspired Systems: Computational and Ambient Intelligence: 10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Part I. Lecture Notes in Computer Science, Volumen 5517. Springer, 2009. 472-479.

DOI: http://dx.doi.org/10.1007/978-3-642-02478-8_59

PNEPs, NEPs for context free parsing: application to natural language processing *

Alfonso Ortega[1], Emilio del Rosal[2], Diana Pérez[1], Robert Mercaş[3], Alexander Perekrestenko[3], Manuel Alfonseca[1]

[1] Dep. Ing. Inf. EPS Universidad Autónoma de Madrid, Spain,
alfonso.ortega@uam.es, diana.perez@uam.es, manuel.alfonseca@uam.es,
[2] Escuela Politécnica Superior Universidad San Pablo C.E.U, Spain
emilio.rosalgarcia@ceu.es
[3] GRLMC, Rovira i Virgili University, Dept. of Romance Filology, Tarragona, Spain
robertgeorge.mercas@estudiants.urv.cat,
alexander.perekrestenko@estudiants.urv.cat

Abstract. This work tests the suitability of NEPs to parse languages. We propose PNEP, a simple extension to NEP, and a procedure to translate a grammar into a PNEP that recognizes the same language. These parsers based on NEPs do not impose any additional constrain to the structure of the grammar, which can contain all kinds of recursive, lambda or ambiguous rules. This flexibility makes this procedure specially suited for Natural Languge Processing (NLP). In a first proof with a simplified English grammar, we got a performance (a linear time complexity) similar to that of the most popular syntactic parsers in the NLP area (Early and its derivatives). All the possible derivations for ambiguous grammars were generated.

1 NEPs for natural language processing: analysis

Computational Linguistics researches linguistic phenomena that occur in digital data. Natural Language Processing (NLP) is a subfield of Computational Linguistics that focuses on building automatic systems able to interpret or generate information written in natural language [1]. This is a broad area which poses a number of challenges, both for theory and for applications.

Machine Translation was the first NLP application in the fifties [2]. In general, the main problem found in all cases is the inherent ambiguity of the language [3].

A typical NLP system has to cover several linguistic levels:

- **Phonological**: Sound processing to detect expression units in speech.
- **Morphological**: Extracting information about words, such as their part of speech and morphological characteristics [4][5]. Best systems has an accuracy of 97% in this level [6].

* This work was partially supported by MEC, project TIN2008-02081/TIN and by DGUI CAM/UAM, project CCG08-UAM/TIC-4425

– **Syntactical**: Using parsers to detect valid structures in the sentences, usually in terms of a certain grammar. One of the most efficient algorithms is the one described by Earley and its derivatives [7,8,9]. It provides parsing in polynomial time, with respect to the length of the input (linear in the average case; n^2 and n^3, respectively, for unambiguous and ambiguous grammars in the worst case) This paper is focused on this step.
– **Semantic**: Finding the most suitable knowledge formalism to represent the meaning of the text.
– **Pragmatic**: Interpreting the meaning of the sentence in a context which makes it possibe to react accordingly.

The two last levels are still far from being solved [10]. Typical NLP systems usually cover the linguistic levels previously described in the following way:

$$\Rightarrow \text{Morphologycal analysis} \Rightarrow \text{Syntax analysis} \Rightarrow \text{Semantic interpretation} \Rightarrow \text{Discourse text processing} \Rightarrow \text{OCR/Tokenization}$$

A computational model that can be applied to NLP tasks is a network of evolutionary processors (NEPs). NEP as a generating device was first introduced in [11] and [12]. The topic is further investigated in [13], while further different variants of the generating machine are introduced and analyzed in [14-18].

In [19], a first attempt was made to apply NEPs for syntactic NLP parsing. Our paper focuses the same goal: testing the suitability of NEPs to tackle this task. We have previously mentioned some performance characteristics of one of the most popular families of NLP parsers (those based on Early's algorithm). We will conclude that our approach has a similar complexity.

While [19] outlines a bottom up approach to natural language parsing with NEPs, in the present paper we suggest a top-down strategy and show its possible use in a practical application.

2 Introduction to jNEP

The jNEP [20] Java written program, freely available at http://jnep.edelrosal.net, can simulate almost all NEPs in the literature. The software has been developed under three main principles: 1) it rigorously complies with the formal definitions found in the literature; 2) it serves as a general tool, by allowing the use of the different NEP variants and can easily be adapted to possible future extensions of the NEP concept; 3) it exploits the inherent parallel/distributed nature of NEPs.

jNEP consists of three main clases (*NEP*, *EvoluionaryProcessor* and *Word*), and three Java interfaces (StoppingCondition, *Filter* and *EvolutionaryRule*)

The design of the NEP class mimics the NEP model definition. In jNEP, a NEP is composed of evolutionary processors and an underlying graph (attribute *edges*), used to define the net topology.

The *NEP* class coordinates the main dynamic of the computation and manages the processors (instances of the *EvolutionaryProcessor* class), forcing them

to perform alternate evolutionary and communication steps. Furthermore, the computation is stoped whenever this is needed.

The Java interfaces are used for those components which more frequently change between different NEP variants: *StoppingCondition*, *Filter* and *EvolutionaryRule*. jNEP implements a wide set of these three components. More can easily be added in the future.

Currently *jNEP* has two lists of choices for the selection of the mode in which it runs: parallel or distributed. For the first list, concurrency is implemented by means of *Threads* and *Processes*, while in the second one, the supported platforms are standard JVM and clusters of computers (by means of JavaParty).

Depending on the operating system, the Java Virtual Machine used and the concurrency option chosen, jNEP will work in a slightly different manner. The users should select the best combination for their needs.

jNEP reads the definition of the NEP from an XML configuration file that contains special tags for any relevant component in the NEP (alphabet, stopping conditions, the complete graph, each edge, each evolutionary processor with its rules, filters and initial contents).

Although some fragments of these files will be shown in this paper, all the configuration files mentioned here can be found at (http://jnep.e-delrosal.net). Despite the complexity of these XML files, the interested reader can see that the tags and their attributes have self-explaining names and values. The reader may refer to the *jNEP user guide* for further detailed information.

3 Top down parsing with NEPs and jNEP

Other authors have previously studied the relationships between NEPs, regular, context-free, and recursively enumerable languages [14-18]. [21] shows how NEPs simulate the application of context free rules ($A \to \alpha, A \in V, \alpha \in V^*$ for alphabet V): a set of additional nodes is needed to implement a rather complex technique to rotate the string and locate A in one of the string ends, then delete it and adding all the symbols in α. PNEPs use context free rules rather than classic substitution ($A \to B, A, B \in V$), as well as insertion and deletion NEP rules. In this way, the expressive power of NEP processors is bounded, while providing a more natural and comfortable way to describe the parsed language for practical purposes.

PNEPs implement a top down parser for context free grammars. The parser is able to generate all the possible derivations of each string in the language generated by the grammar. Its temporal complexity is bounded by the length of the analyzed string. This bound can be used to stop the computation when processing incorrect strings, thus avoiding running the PNEP for a possible infinite number of steps.

The PNEP is built from the grammar in the following way: (1) We assume that each derivation rule in the grammar has a unique index that can be used to reconstruct the derivation tree. (2) There is a node for each non terminal. Each node applies to the strings all the derivation rules for its non terminal.

The filters, as well as the graph layout, allow all the nodes to share all the intermediate steps in the derivation process. (3) There is an additional output node, in which the *parsed string* can be found: this is a version of the input, enriched with information that will make it possible to reconstruct the derivation tree (the rules indices). (4) The graph is complete.

Obviously the same task can be performed using a trivial PNEP with only a node for all the derivation rules. However, the proposed PNEP is easier to analyze and more usefull to distribute the work among several nodes.

We shall consider as an example the grammar $G_{a^n b^n c^m}$ induced by the following derivation rules (notice that indexes have been added in front of the corresponding right hand side):

$$X \Rightarrow (1)SO \ , \ S \Rightarrow (2)aSb|(3)ab \ , \ O \Rightarrow (4)Oo|(5)oO|(6)o$$

It is easy to prove that the language corresponding to this grammar is $\{a^n b^n o^m \mid n, m > 0\}$. Furthermore, the grammar is ambiguous, since every sequence of o symbols can be generated in two different ways: by producing the new terminal o with rule 4 or with rule 6.

The input filters of the output node describe parsed copies of the initial string. In other words, strings whose symbols are preceded by strings of any length (including 0) of the possible rules indexes. As an example, a parsed version of the string $aabboo$ would be $12a3abb5o6o$.

We will assume in this paper that a PNEP Γ is formally defined as follows:

$\Gamma = (V, N_1, N_2, ..., N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n, N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

- M_i is a finite set of context-free evolution rules
- A_i is the set of initial strings in the i-th node.
- PI_i and PO_i are subsets of V^* representing respectively the input and the output filters. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i (w \in PO_i)$. In this paper we will use two kind of filters:
  - Those defined as two components (P, F) of *P*ermitting and *F*orbidding contexts (a word w passes the filter if $(\alpha(w) \subseteq P) \wedge (F \cap \alpha(w) = \Phi)$).
  - Those defined as regular expressions r (a word w passes the filter if $w \in L(r)$, where $L(r)$ stands for the language defined by the regular expression r).

Finally, $G = (N_1, N_2, ..., N_n, E)$ is an undirected graph (whose edges are E), called the underlying graph of the network.

We will now describe the way in which our PNEP is defined, starting from a certain grammar. Given the context free grammar $G = \{\Sigma_T = \{t_1, ..., t_n\}, \Sigma_N = \{N_1, ..., N_m\}, A, P\}$ with $A \in \Sigma_N$ its axiom and $P = \{l_j \rightarrow \gamma_j \mid j \in \{1, .., k\}, l_j \in \Sigma_N \wedge \gamma_j \in (\Sigma_T \cup \Sigma_N)^*\}$ its set of k production rules the PNEP is defined as

$$\Gamma_G = (V = \Sigma_T \cup \Sigma_N \cup \{1, ..., k\}, node_{output}, N_1, N_2, ..., N_m, G)$$

where (1) $node_{output}$ is the output node; (2) G is a complete graph and (3) the *input node* A, the only one with a non empty initial content (A). Each non terminal node N_i in the PNEP has a substitution rule for each derivation rule in the grammar applicable to it. This rule changes the nonterminal by a string made by uppending the right hand side of the derivation rule with the index of the rule in P. For example, the PNEP for grammar $G_{a^n b^n o^m}$ described above has a node for nonterminal S with the following substitution rules: $\{S \rightarrow 2aSb, S \rightarrow 3ab\}$ The input filters of these nodes allow all strings containing some copy of their non terminal to input the node. Strings that do not contain a copy of the non terminal pass the output filter. We can get this behavior by using the set with the non terminal symbol of the node ($\{N_i\}$) both as the permitted input filter and as the forbidden output filter.

The input filter for the output node $node_{output}$ has to describe what we have called *parsed strings*. Parsed strings will contain numbers, corresponding to the derivation rules which have been applied, among the symbols of the initial string. We can easily create a regular expression and define the input filter by means of membership. For example, in order to parse the string *aabbo* with the grammar we are using above, the regular expression can be $\{1,2,3,4,5,6\} * a\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*o$. Our PNEP will stop computing whenever a string enters the output node.

The complete PNEP for our example ($\Gamma_{a^n b^n o^m}$) is defined as follows:

- Alphabet $V = \{X, O, S, a, b, o, 1, 2, 3, 4, 5, 6\}$
- Nodes
  - $node_{output}$: $A_{output} = \Phi$ is the initial content; $M_{output} = \Phi$ is the set of rules; $PI_{output} = \{$ (regular expression membership filter); $\{\{1,2,3,4,5,6\} * a\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*o\}\}$; $PO_{output} = \Phi$ is the output filter
  - N_X: $A_X = \{X\}$; $M_X = \{X \rightarrow 1SO\}$; $PI_X = \{P = \{X\}, F = \Phi\}$; $PO_X = \{F = \{X\}, P = \Phi\}$
  - N_S: $A_S = \Phi$; $M_S = \{S \rightarrow 2aSb, S \rightarrow 3ab\}$; $PI_S = \{P = \{S\}, F = \Phi\}$; $PO_S = \{F = \{S\}, P = \Phi\}$
  - N_O: $A_O = \Phi$; $M_O = \{O \rightarrow 4oO, O \rightarrow 5Oo, O \rightarrow 5o\}$; $PI_O = \{P = \{O\}, F = \Phi\}$; $PO_O = \{F = \{O\}, P = \Phi\}$
  - It has a complete graph
  - It stops the computation when some string enters $node_{output}$

Some of the strings generated by all the nodes of the PNEP in succesive communication steps when parsing the string *aboo* are shown below (each set corresponds to a different step): $\{X\} \Rightarrow \{1SO\} \Rightarrow \{..., 13abO, ...\} \Rightarrow$
$\{..., 13ab4Oo, 13ab5oO, ...\} \Rightarrow \{..., 13ab46oo, ..., 13ab5o6o, ...\}$

The last set contains two different derivations for *aboo* by ($G_{a^n b^n n o^m}$), that can enter the output node and stop the computation of the PNEP.

It is easy to reconstruct the derivation tree from the parsed strings in the output node, by following their sequence of numbers. For example, consider the parsed string 13*ab*6*o* and its sequence of indexes 136; *abo* is generated in the following steps: $X \Rightarrow$ (rule 1 $X \Rightarrow SO$) SO , $SO \Rightarrow$ (rule 3 $S \Rightarrow ab$) abO , $abO \Rightarrow$ (rule 6 $O \Rightarrow o$) abo

In section 2 we have described the structure of the xml input files for *jNEP*. Two the sections of the xml representation of the NEP $\Gamma_{a^n b^n o^m}$ (the output node and the node for axiom X) are shown below.

```
<NODE initCond="">
   <EVOLUTIONARY_RULES>
   </EVOLUTIONARY_RULES>
   <FILTERS>
     <INPUT type="RegularLangMembershipFilter" regularExpression="[1-6]*a[1-6]*b[1-6]*o[1-6]*o"/>
     <OUTPUT type="1" permittingContext="" forbiddingContext=""/>
   </FILTERS>
 </NODE>

 <NODE initCond="X">
   <EVOLUTIONARY_RULES>
     <RULE ruleType="substitution" actionType="ANY" symbol="X" newSymbol="1_S_O"/>
   </EVOLUTIONARY_RULES>
   <FILTERS>
     <INPUT type="1" permittingContext="X" forbiddingContext=""/>
     <OUTPUT type="1" permittingContext="" forbiddingContext="X"/>
   </FILTERS>
 </NODE>
```

The nodes for other non terminal symbols are similar, but with an empty ("") *initial condition* and their corresponding derivation rules.

4 An example for natural language processing

As described in section 1, the complexity of the grammars used for syntactic parsing depends on the desired target. These grammars are usually very complex, which makes them one of the bottlenecks in NLP tasks.

We will use the grammar deduced from the following derivation rules (whose axiom is the non terminal Sentence). This grammar is similar to those devised by other authours in previous attempts to use NEPs for parsing (natural) languages [19]. We have added the index of the derivation rules, that will be used later.

Sentence	→ (1)	NounFraseStandard PredicateStandard
	| (2)	NounFrase3Singular Predicate3Singular
NounFrase3Singular	→ (3)	DeterminantAn VowelNounSingular
	| (4)	DeterminantSingular NounSingular
	| (5)	Pronoun3Singular
NounFraseStandard	→ (6)	DeterminantPlural NounPlural
	| (7)	PronounNo3Singular
NounFrase	→ (8)	NounFrase3Singular | (9) NounFraseStandard
PredicateStandard	→ (10)	VerbStandard NounFrase
Predicate3Singular	→ (11)	Verb3Singular NounFrase
DeterminantSingular	→ (12)	a | (13) the | (14) this
DeterminantAn	→ (15)	an
VowelNounSingular	→ (16)	apple
NounSingular	→ (17)	boy
Pronoun3Singular	→ (18)	he | (19) she | (20) it
DeterminantPlural	→ (21)	the | (22) several | (23) these
NounPlural	→ (24)	apples | (25) boys
PronounNo3Singular	→ (26)	I | (27) you | (28) we | (29) they
VerbStandard	→ (30)	eat
Verb3Singular	→ (31)	eats

It is worth noticing that this grammar is very simple. As described in section 1, NLP syntax parsing usually takes as input the results of the morphological analysis. In this way, the previous grammar can be simplified by removing

the derivation rules for the last 9 non terminals (from DeterminantSingular to Verb3Singular): those symbols become terminals for the new grammar.

Notice, also, that this grammar implements grammatical agreement by means of context free rules. For each non terminal, we had to use several different *specialized versions*. For instance, NounFraseStandard and NounFrase3Singular are specialized versions of non terminal NounFrase. These rules increases the complexity of the grammar.

We can build the PNEP associated with this context free grammar by following the steps described in section 3.

Let us consider the English sentence *the boy eats an apple* Some of the strings generated by the nodes of the PNEP in succesive communication steps while parsing this string are shown below (we show the initials, rather than the full name of the symbols). A left derivation of the string is highlighted: { S } ⇒ { ..., 2 NF3S P3S, ... } ⇒ { ..., 2 4 DS NS P3S, ... } ⇒ { ..., 2 4 13 the NS P3S, ... } ⇒ { ..., 2 4 13 the 17 boy P3S, ... } ⇒ { ..., 2 4 13 the 17 boy 11 V3S NF, ... } ⇒ { ..., 2 4 13 the 17 boy 11 31 eats NF, ... } ⇒ { ..., 2 4 13 the 17 boy 11 31 eats 8 NF3S, ... } ⇒ { ..., 2 4 13 the 17 boy 11 31 eats 8 3 DA VNS, ... } ⇒ { ..., 2 4 13 the 17 boy 11 31 eats 8 3 15 an VNS, ... } ⇒ { ..., 2 4 13 the 17 boy 11 31 eats 8 3 15 an 16 apple, ... }

If we analyze now an incorrect sentence, such as *the boy eat the apple*, the PNEP will continue the computation after the steps summarized above, because in this case it is impossible to find a parsed string. It is easy to modify our PNEP to stop when this circumstance happens.

5 Conclusions and further research lines

We have found that a very simple and straightforward use of NEPs, associated to context-free grammars, results in a parsing technique with similar characteristics to the most popular syntactic parsers in the NLP area. PNEPs seem to be an appropiate and natural approach to tackle both parsing of context-free languages and the syntactic processing of natural languages.

In the future we plan to use a more realistic grammar for NLP purposes; to incorporate this technique into a linguistic corpus in which we are currently working; to test our approach with realistic NLP benchmarks; and to explore the possible design of compiler tools based on PNEPs.

References

1. Volk, M. (2004), Introduction to Natural Language Processing, Course CMSC 723 / LING 645 in the Stockholm University, Sweden.
2. Weaver, W. (55), Translation, Machine Translation of Languages: Fourteen Essays, 15–23.
3. Mitkov, R. (2003), The Oxford Handbook of Computational Linguistics, Oxford University Press.
4. Mikheev, A. (2002), Periods, capitalized words, etc., Computational Linguistics 28(3), 289–318.

5. Alfonseca, E. (2003), An Approach for Automatic Generation of on-line Information Systems based on the Integration of Natural Language Processing and Adaptive Hypermedia techniques, PhD thesis, Computer Science Department, UAM.
6. Brants, T. (2000), TnT–a statistical part-of-speech tagger, Proceedings of the 6th Conference on Applied Natural Language Processing, 224–231.
7. Earley, J. (1970), An efficient context-free parsing algorithm, Communications of the ACM 13(2), 94–102.
8. Seifert, S. & Fischer, I. (2004), Parsing String Generating Hypergraph Grammars, Springer, p.352-367.
9. Zollmann, A. & Venugopal, A. (2006), Syntax augmented machine translation via chart parsing, in Proccedings of the Workshop on Statistic Machine Translation, HLT/NAACL, New York, June.
10. Gomez, C.; Javier, F.; Valle Agudo, D.; Rivero Espinosa, J. & Cuadra Fernandez, D. (2008), Methodological approach for pragmatic annotation, Procesamiento del lenguaje Natural, 209–216.
11. Csuhaj-Varju, E. and Salomaa, A. (1997): Networks of parallel language processors. Lecture Notes on Computer Science 1218.
12. Csuhaj-Varju, E. and Mitrana, V. (2000). Evolutionary systems:a language generating device inspired by evolving communities of cells. Acta Informatica 36.
13. Castellanos, J., Martin-Vide, C., Mitrana, V., and Sempere, J. M. (2001): Solving np-complete problems with networks of evolutionary processors. Lecture Notes in Computer Science, IWANN(2048):621-628.
14. Castellanos, J., Leupold, P., and Mitrana, V. (2005): On the size complexity of HNEPs. Theoretical Computer Science, 330(2):205-220.
15. Manea, F. (2004): Using ahneps in the recognition of context-free languages. In Proceedings of the Workshop on Symbolic Networks, ECAI 2004.
16. Manea, F. and Mitrana, V. (2007): All np-problems can be solved in polynomial time by AHNEPs of constant size. Inf. Process. Lett., 103(3):112-118.
17. Margenstern, M., Mitrana, V., and Perez-Jimenez, M. (2004): Accepting hybrid networks of evolutionary processors. In Pre-proceedings of DNA 10, pages 107-117.
18. Martin-Vide, C., Mitrana, V., Perez-Jimenez, M., and Sancho-Caparrini, F. (2003): Hybrid networks of evolutionary processors. In Proc.GECCO, LNCS 2723, pages 401-412, Berlin. Springer.
19. Bel Enguix, G., Jimenez-Lopez, M. D., Mercaş, R. and Perekrestenko, A. (2009): Networks of evolutionary processors as natural language parsers. In proc. ICAART.
20. E. del, Nuez, R., Castaeda, C., Ortega, A., 2008. Simulating NEPs in a cluster with jNEP. In Proceedings of International Conference on Computers, Communications and Control, ICCCC 2008,
21. Csuhaj-Varju, E. Martin Vide, C., Mitrana, V.: HNEPs are Computationally Complete. Acta Informatica