**CREATING AND PROCESSING A CORPUS**

Prihantoro

Universitas Diponegoro

prihantoro2001@yahoo.com

**Abstract**

This paper seeks to describe some crucial importance of corpus and text processing. Corpus is is a projection of how language is used by its speakers. Technology support has improved corpus for easier maintenance, made it space-saving, and it may electronically structure its data. The latest offers much freedom for corpus users to access and exploit it for language teaching, analysis or other specified tasks. This paper will demonstrate how to use open-access corpus on internet such as Corpus of Contemporary American English (COCA) and British National Corpus (BNC). Besides how to use a corpus, another crucial importance that this paper seeks to describe is how to build a corpus. In this paper, the writer will use UNITEX, a corpus (text-based) processing software. This software will demonstrate steps of corpus building, ranging from text collection, annotation, electronic dictionary application to some natural language based operations ranging from pattern matching, concordance, to simple extraction. It will show how graph technology may outperform regular expression, a retrieval method exploited by other corpus processor, in terms of writing output.

*Keywords: Corpus, Corpus Processing Software, Concordance, Pattern matching.*

**INTRODUCTION**

*Corpora* (the plural form of *corpus*) is considered equal to some extent as linguistic data. It is proper to deduce that a corpus contain linguistic data. However, not all linguistic data can be labeled as corpora, or corpus. These days, the term *corpus* usually refers to a given structured linguistic data, where the collection of texts, or spoken data, is transcribed digitally into machine readable format. See Richards & Schmidt (2002)

The conversion of linguistic data to computer readable format is essential to ease the preservation and development of the data. To be open and accessible are two significant features of any corpora. When adhering to these features, corpora provide more support for developing and maintaining machine readable linguistic data. The users can employ the data for different purposes, such as purely linguistic research, or educational purpose, even across geographical border.

Widely existing corpora are usually maintained and developed by linguists with the support of computational knowledge. In this research, some of the corpora on webs are discussed such as : National Corpus[[1]](#footnote-2) (BNC), Collin Corpus[[2]](#footnote-3) and the Bank of English, the international corpus of English,[[3]](#footnote-4) the Corpus of Contemporary American English (COCA).

Numerous linguists across the world have precious linguistic data, but are lacking of support to manage and process their linguistic data as a corpus. Only with proper text management and processing, we can classify linguistic data as a corpus. To overcome this problem, linguists can either study programming, or use the support a person with computational capability. Another proposal is to use user-friendly corpus processing software to manage and process their personal corpus before sharing it on-line. This software will introduce them to the basic skills of computation, especially skills related to computational linguistics and natural language processing.

Besides demonstrating how linguists can benefit from corpora on webs, this paper is aimed at demonstrating how some corpus processing tasks can be performed with the support of user-friendly corpus processing software, and without exhaustive programming training. The writer uses UNITEX (Paumier, 2008), a corpus processing software with local grammar platform (Gross, 1993, 1994, 1997), which are also used in INTEX (Silberztein, 1993) and NOOJ. Local Grammar platform have been used in the research of various languages such as French (Silberztein, 1993), English (Gross, 1999 and Prihantoro 2011c), Indonesian (Prihantoro, 2011a & 2011b), Korean (Nam & Choi, 1997 and Prihantoro, 2011a), and Arabic (Traboulsi, 2009).

After the introduction, this paper briefly introduces some of the corpora on webs, with particular emphasis on automatic retrieval by using regular expressions in the Corpus of Contemporary American English (COCA). Regular expressions are useful to retrieve linguistic data that they require. With regular expressions, users can posit constraints to fit their objectives. In the succeeding chapter of this paper, UNITEX shows how to perform more powerful retrieval to match patterns beyond word level by using Local Grammar Graphs (LGGs). Therefore, users can design custom made retrievals beyond the default (pre-programmed) ones as some default commands on corpora on web.

**TOOLS AND METHODOLOGY**

***Corpora on Webs: Retrieval with Regular Expressions***

Corpus data on webs are usually fixed; the data can only be modified by its creator, manager, or users with special authorization. However, they are open and accessible for users to navigate. Some frequently accessed corpora such as British National Corpus (BNC) and the Corpus of Contemporary American English (COCA) are freely accessible, and even if you are requested to register, they are free of charge. Browsing these corpora, users are equipped with a query box (like google), where the queries (word or phrase) are typed. Figure 1 and 2 present the index page of BNC for *design* query:

**Figure 1. Query Box**



**Figure 2. Usage and Source**

****

****

Illustration 1 is quite common, presuming that we are all familiar in using search engines such as Google, Yahoo or some other search engines available on internet. Illustration 2 (upper figure) presents the result of the search, where it summarize the search into top 50. The figure on the lower side indicates the source or the corpus where the token is used. .

Most of the corpora available on line make use of regular expression in order to perform automatic retrieval. Regular expression can be powerful when the corpus is already annotated. See table 1:

**Table 1. Some Regular Expressions for Corpus of the Contemporary American English (COCA)**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Regular expression list | Keyword | Retrieval Result  |
| 1 | Strings of letters only | Ship | Purely character base |
| 2 | Square brackets around letter string | [go][ship] | Retrieve all possible inflected and derived forms of ‘go’ such as *went*, *gone* and *going*, ‘ship’ as a noun and text as a verb |
| 3 | Question mark before/after string of letters | prove? | Retrieve all strings of ‘prove’ plus other character/s, such as: *proves*, *proved* and *proving*. But it cannot retrieve the derived form such as proof |
| 4 | Star (\*) and space before or after string/strings of letters | Curiosity killed the \* | Retrieve all possible tokens after the query. In this case it possibly retrieves, *curiosity killed the* ***cat*** |

List of regular expression commands on table 1 applies for COCA. When one of these commands is executed it will display the result and the context where the token is used. When [go] command is executed, it is most likely that users obtain all possible forms of *go* such as *went, gone, going* and *goes*. Consider figure 2:

**Figure 2. Possible Forms of [go]**

****

The presentation on figure 2 demonstrates the execution of command [go] on the corpus, and notice that the possible forms are presented. When one of the possible forms is selected, it is possible to observe how a given form is used in context, as presented by figure 3 for *going*:

**Figure 3. Contextual Use of *going***

****

Ambiguity is classical problem encountered in automatic retrieval. COCA has some functionality to overcome this problem. In the study of automatic retrieval, when the coverage of the retrieval is too large, the result is often less accurate. One of the solutions is to narrow down the search and get small, but more concise result. In resolving ambiguities, we can interpret this as setting constraint to the retrieval. The retrieval of the word <text> is most likely to provide, at least, two types of tokens: text as a verb and a noun. Assuming that users want to obtain only the verb, *text* as nouns must be excluded. It suggests a necessity to set constraint on the retrieval of *text* to include only the verb. Consider figure 4:

**Figure 4. Setting Constraint of *ship* as a Verb**



There is another additional command d to write, which is [v\*] after the token *ship*. This command has set one constraint where it limits the retrieval only for the token of *text* as a verb. Consider the lower part of figure 4.

So far, regular expression is effective to retrieve tokens on the word level even with constraints. However, sometimes the target string has the complexity beyond word level. The next section discusses how Local Grammar is proposed as one of the solutions for this problem.

**Local Grammar Graphs: Beyond Regular Expressions**

In the preceding section, I have demonstrated how regular expression can be used to perform more powerful research beyond character based retrieval. The retrieval can successfully identify and recognize words by setting various constraints such as collocation, grammatical class, possible derivation and inflection.

A new challenge is to perform retrieval tasks beyond word level. In this paper, I propose a method of automatic retrieval named Local Grammar proposed by Maurice Gross, which becomes the platform of some corpus processing software such as: INTEX (Silberztein, 1993), NOOJ and UNITEX (Paumier, 2008). Local grammar itself can be described as finite state grammars or finite state automata which express strings of a natural language. Silberztein (1993) has designed user friendly interface graph for local grammar and it is adopted in those three software under the name of Local Grammar Graph (LGG). Consider figure 5:

**Figure 5. Prepositional Phrase Formula and Its Retrieval Result**



Figure 5 has illustrated the LGG for the retrieval of prepositional phrase and the result of the retrieval as well. Note that each code complies with annotation code written in the lexical resource of UNITEX. The code <PREP> stands for preposition, <DET> for determiner and <N> for noun, respectively. The result, are shown in concordance by alphabetical order.

When more specific retrieval is required, constraints are established. For instance, the constraint limits the determiners to definite and indefinite article (‘a’ and ‘the’). There are two methods for designing the formula. One is to write the annotation codes, another is to write the terminal (vocabulary entry) as illustrated by figure 6:

**Figure 6. Setting Constraints for Prepositional Phrase and Its Result**





Two formulas are presented by figure 6. Code <DET> expresses determiner. As many determiners exist, we might set constraint that fit the target string. There are two constraints that have been established: definite and indefinite article. Constraint for definite article is expressed by the annotation code <Ddef>, while constraint for indefinite article is expressed by the code <Dind>. Instead of writing these codes, the terminals (‘a’ and ‘the’) can be written direct. These two formulas lead to the same result as illustrated by the lower figure of figure 6. The formula on figure 6 sets grammatical constraints. But sometimes problems pose when semantic constraint is also required. For instance, the target string is noun with animal property [+ANL] . Consider figure 7:

**Figure 7. Semantic Constraint for Nouns that are Animal**





This suggests that there are various different methods for automatic retrieval. We, however, must select which method that fits the purpose. LGG is required when a set of complex constraints needs to establish, especially to retrieve strings beyond word level. But when the retrieval is simpler, such as character based, or with less constraint, regular expression is already sufficient.

When a formula is written, the codes (grammar or semantic) refer to how the lexicon is annotated in the lexical resource. The formula works in condition that the lexical resource is already applied to the corpus. This suggests that data annotation in the lexical resource is important. Therefore, we are required to understand how the data in the lexical resource is annotated. The annotation process is described in the next section of this paper.

**DEMONSTRATION AND DISCUSSION**

This section describes steps of how to build and process your corpus. Corpus is collection of documents; the documents are available in many different format. In *data upload*, we will discuss on issues related to the conversion of existing documents (PDF, PHP, HTML, JPEG and etc) to a corpus processor readable format. *Data Annotation* is the part where we discuss the annotation of the corpus before further processing. The next sub section deals with the processing of the corpus after annotation. It will show how in Unitex is capable of, not only performing pattern recognition, but also writing an output to the recognized sequence.

***Data Upload***

The first step to build a corpus is to make the document machine-readable. Machine-readable, here, does not mean that any computer readable format can be processed by a corpus processing software. Machine readable refers to format(s) that can be managed by a corpus processing software. As for most corpus processor, plain-text is the most friendly format that is available for processing. Documents in PDF, Microsoft Word, or other text processor must be converted to plain text file (.txt).

Figure 8. Ivanhoe in PDF format

****

Figure xxx. Ivanhoe in HTML format

****

This applies also to texts available on-line such as in php or html format. Conversion of these formats to plain text format is simple. You can do it with freely available file converter or do a simple copy-paste to plain text program. In Microsoft windows, plain text is created by notepad. Go to start > All Program > Accessories > Notepad. Copy the text and paste to notepad and save. If your computer does not have plain text writer, you can download notepad to your computer.

**Figure 9. Ivanhoe in JPEG/Hardcopy Format**



The most challenging conversion is from hard copy or photo format to plain text. Sometimes we find documents in hard copy, or it is soft copy but in photo format. Some PDF format allows photo scans as well. What we have to do is ‘read’ the documents, either in hard copy or photo format, by optical character recognition (OCR) scanner. When a scanner is equipped with OCR, it does not just scan the document, but it also ‘reads’ the characters and encode it to the output file. This is important as corpus processor only process characters and scripts, not picture or photo. If the document is already in JPEG or other photo format file, you can install OCR or OCR-conceived software such as Nitro PDF or Professional PDF.

***Data Annotation***

Text processed UNITEX is annotated by using an entry-line formalism based lexical resource. In this formalism, annotation code(s) are written after the entry. The code ranges from grammar, semantic and inflection[[4]](#footnote-5) code. Consider the codes for some entries, written in the following entry lines from (Paumier, 2008):

1. aviatrix,N4+Hum
2. matrix,N4+Math
3. radix,N4

This entry is very simple. It writes only one grammar code <N>, which stands for a noun. Therefore, all the entries are classified as nouns. A number after the first grammar code express the inflectional group. This means that the three entries take LGG number 4, which is a morphological graph for plural inflection.

**Figure 10. Inflection LGG 4**



When an entry is inflected by LGG 4, the system creates the inflected forms. According to figure 4, there are two inflected forms. The upper line does not modify anything on the entry. For instance, *matrix* is inflected as *matrix* with code <s> standing for singular. The lower inflectional line, however, deleted one character from the left-most of the entry [*matrix --> matri*], and concatenate *ces* [*matri* --> *matrices*]. It also labels the inflected form with <p> as plural. Therefore, this inflectional graph, results on two inflected forms: singular and plural form of *matrix*: [matrix] and [matrices. These completed graphs are compiled and inflected to the lexical resource. As a consequence, more entry lines are created in the lexical resource as the result of the inflection graphs. Consider the figure 11, which is the result of the applied compilation graphs:

**Figure 11. Entry Lines for Inflected form Lexical Resource**

****



Consider entry line 223085 <matrix> and entry line 223046 <matrices, matrix>. Entry line 223046 consists of matrices and matrix. This indicates that matrices, is the inflected form of matrix that undergo modification. I must inform you that this LGG is for plural only, and there are also another LGG that we design for other inflection, even for other plural methods. For instance *ox* and *oxen*, *mouse* or *mice* and etc. When the completed inflectional LGGs are inflected to the lexical resource, a new and richer lexical resource is created. This new lexical resource is applied to a custom made corpus that each user creates.

***Applying Lexical Resource to Custom-made Corpus***

As one of the methods for automatic retrieval, LGG can apply only when the existing corpus data is annotated (with the lexical resource). UNITEX allows the creation of custom made corpus. The first step is to upload the text that is in computer readable form. Here, we need to convert all text formats to <.txt> extension, the type of format that complies with UNITEX. The next step is to open the text. Remember that the text has not been annotated, so the option that we must choose is *open* as presented by figure 12 (Paumier, 2008):

**Figure 12. Opening raw text**



The next step after the text is open is applying the lexical resource to the raw corpus text, also known as the stage of preprocessing. The aim is to tag (annotate) all the tokens by using the existing lexical resource which has been inflected by various inflectional LGGs. Consider figure 13 from UNITEX (Paumier, 2008):

**Figure 13. Preprocessing and Lexical Parsing**



This process annotates all the tokens, by using the existing lexical resource. After the application of the lexical resource, UNITEX displays the annotated text with wordlist and statistical information such as frequency of existing tokens, the percentage and etc as presented by figure 14:

**Figure 14. Tagged Text**



When the text is already annotated, UNITEX gives liberty for its users to apply automatic retrieval by either using regular expression, or LGGs. The result, as it has been commented previously, is displayed as a concordance file. Consider figure 15 from UNITEX (Paumier, 2008):

**Figure 15. Locate Pattern**



In Locate Pattern box, there are two automatic retrieval methods that to select: regular expression, or graph (LGG). The strength of LGG based retrieval has been commented on the previous section of this paper. However, there is one more advantage by using LGG based retrieval, even for a very simple locate pattern, such as one word search. Regular expression allows recognition of target strings on the basis of the formula that users have typed on the searching box. However, by using LGG search, output to recognized sequence can be generated. Grammar outputs in the locate pattern box allows you to show the output <*merge with the input text*> or use the output to replace the recognized strings <*replace recognized sequences*> as it is presented by figure 16:

**Figure 16. LGG with Output**



Figure 14 is divided into upper and lower part. The upper part of this figure indicates the LGG that has been defined by the users to choose in the retrieval. Going back to figure 5 on section 3 of this paper, the two LGGs resemble one and each other. Indeed, they are meant to obtain equal result. However, in figure 14, the LGG is equipped with [PP= ] output. It informs you that the strings you recognize is a kind of prepositional phrase in English [PP] by generating output on the concordance.

**CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH**

This paper has illustrated the corpora either on the web, or corpora that user can define (custom made). The proposal of custom made corpora grounds on what users require. Corpora on web are quite representative, but these corpora might fail to notice what some users require. For instance, users want to perform automatic retrieval on today’s newspaper. As for this case, not many corpora update its content every day, if there is none. Custom made corpora is one of the proposed solutions for this problem.

This paper has also demonstrated two automatic retrieval methods: one by regular expressions (mostly used for corpora on webs), and LGG based retrieval (used in Corpus processing software with Local Grammar platform). LGG based retrieval has proven to be more powerful in terms of complexity of the strings that users might want to recognize. Another advantage of using LGG for automatic retrieval is its power to generate output, which regular expressions cannot perform. However, the retrieval is optimum when the corpus text is already annotated with lexical resource. The accuracy of the retrieval crucially depends on the quality of the lexical resource.

Until now, most of the large corpora are dedicated for English. There are some other corpora for the research of other languages well, but they are not as well-known as English corpora, and usually aim on particular research groups. For instance, Malay concordance project for Classic Malay and Indonesian[[5]](#footnote-6) language, Sheffield corpus of Chinese[[6]](#footnote-7), Tanaka Corpus[[7]](#footnote-8) for Japanese, King Sejong Corpus for Korean[[8]](#footnote-9), Quranic Arabic Corpus[[9]](#footnote-10) for Arabic. Notice that from the list of the corpora, some of them are dedicated for languages which are widely spoken (or at least the number of the speakers is significantly high), such as: classic Malay and Indonesian, Chinese and Arabic. However, we notice that corpora of some languages with less speakers than Indonesian, such as Japanese and Korean, do exist. What about contemporary Indonesian corpora? A government institution in Indonesia (BPPT) has already designed a corpus, but it still requires a specific processing software to perform automatic retrieval. Therefore, a suggestion for further research is how to design a specific processing software or use the pre-existing software like UNITEX to perform various language processing tasks on contemporary Indonesian corpus.

# Bibliography

Gross, M. (1999). A Bootsrap method for Constructing Local Grammars. *Contemporary Mathematics: Proceedings of the Symposium* (pp. 229-250). Belgrade, Serbia: University of Belgrade Press.

Gross, M. (1994). Constructing Lexicon Grammars. In A. Zampolli, & B. Atkins, *Computational Approaches to the Lexicon* (pp. 213-263). Oxford: Oxford University Press.

Gross, M. (1984). Lexicon Grammar and the Syntactic Analysis of French. *Proceedings of the 10th International Conference on Computational Linguistics* (pp. 275-282). Stanford: Association for Computational Linguistics.

Gross, M. (1993). Local Grammars and Their Representations by Finite State Automata. In M. Hoey, *Data, Description, Discourse: Papers on the English Language in Honour of John Mc.Sinclair* (pp. 26-38). London: Harper Collins.

Gross, M. (1997). The Construction of Local Grammars. In E. Roche, & Y. Schabes, *Finite State Language Processing* (pp. 329-354). Massachusetts: MIT Press.

Nam, J. S., & Choi, K. S. (1997). A Local Grammar Based Approach to Recognizing Proper Names in Korean Texts. *Workshop on Very Large Corpora: ACL*, (pp. 273-288). Hongkong.

Paumier, S. (2008). *Unitex Manual.* Paris: Universite Paris Est Marne La Valee & LADL.

Prihantoro. (2011a). *A Comparative Study of Korean and Indonesian Noun Phrases with Numerals: Building a Machine Readable Dictionary (M.A Thesis).* Seoul: Hankuk University of Foreign Studies.

Prihantoro. (2011b). Local Grammar Based Auto Prefixing Model for Automatic Extraction in Indonesian Corpus (Focus on Prefix MeN-). *Proceedings of International Congress of Indonesian Linguists Society (KIMLI)* (pp. 32-38). Bandung: Universitas Pendidikan Indonesia Press.

Prihantoro. (2011c). Transducer for Auto-Convert of Archaic to Present Day English: A Support for Computer Assisted Language Learning. *Proceeding for The 16th English in Southeast Asia Conference.* Yogyakarta: Sanata Dharma University Press.

Richards, J.-C., & Schmidt, R. (2002). *Longman Dictionary of Language Teaching and Applied Linguistics.* New York: Pearson Education.

Silberztein, M. (1993). *Dictionnaires Electroniques et Analyse Automatique de Textes: Le Systeme INTEX.* Paris: Masson.

Silberztein, M. (1997). The Lexical Analysis of Natural Languages. *Finite State Language Processing* (pp. 99-108). Massachusetts: MIT Press.

Traboulsi, H. (2009). Arabic Named Entity Extraction: A Local Grammar Based Approach. *Proceedings of the International Multiconference Science and Information Technology* (pp. 139-143). Mragowo, Poland: Polish Information Processing Society.

1. www.natcorp.ox.ac.uk [↑](#footnote-ref-2)
2. www.collinslanguage.com/collins-elt-learners-of-english/cobuild [↑](#footnote-ref-3)
3. www.ucl.ac.uk/english-usage/projects/ice.htm [↑](#footnote-ref-4)
4. In the corpus processing software that we use in this research, the term inflection refers to both inflection and derivation. [↑](#footnote-ref-5)
5. www.mcp.anu.edu.au/ [↑](#footnote-ref-6)
6. www.hrionline.ac.uk/scc/ [↑](#footnote-ref-7)
7. www.manythings.org/corpus/ [↑](#footnote-ref-8)
8. www.sejong.or.kr/ [↑](#footnote-ref-9)
9. www.corpus.quran.com/ [↑](#footnote-ref-10)