行政院國家科學委員會專題研究計畫 成果報告

設計語法蒐尋機制於英語學習上的應用(第 3 年)
研究成果報告(完整版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 95-2520-S-032-004-MY3
執 行 期 間 ：97 年 08 月 01 日至 98 年 07 月 31 日
執 行 單 位 ：淡江大學資訊工程學系

計 畫 主 持 人 ：郭經華

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 98 年 10 月 30 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

設計語法蒐尋機制於英語學習上的應用

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC95－2520－S－032－004－MY3
執行期間：95 年　08　月　01　日至　98 年　07　月　31　日

計畫主持人：郭經華
共同主持人：
計畫參與人員：陳瑞璟、呂友端、林谷鴻、邱彥翔、黃品諺、
曾彥儒

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：淡江大學

中　華　民　國　98　年　10　月

目錄

1. 前言

本計畫主要目的為設計開發一語法(Syntax)搜尋機制，以輔助教師及學習者在廣大語料中擷取包含特定語法的文句或文件，由於我國教育的第一外語以英文為主，所以我們的應用方向也將著重在英文語料搜尋上。目前大多數的語料庫(Corpus)或搜尋引擎都只提供關鍵字查詢(Keyword Search)，此功能對於學習英文字彙有很大的幫助。但學習英文並不是只有字彙，英文語法的學習也十分重要。目前學習語法的方式不外乎是利用已編輯好的書籍教材進行學習，雖然此法可有組織有效率的學習語法，但此類教材因頁數限制，無法提供大量的例句。而大量的例句讓學習者在學習語法的過程中得知真實環境的用法，如此將可大幅提升學習效率，然而關鍵字查詢無法達到搜尋語法的目的，所以不易幫助學習者提供大量相關語法的例句。例如，如果我們要學習 keep + 現在進行式的用法，如果利用關鍵字查詢搜尋 keep，我們雖然也可以得到如 keep walking, keep going 的例句，但同時也參雜著其他的例句，例如 keep in touch, you can keep this 等，如此無章法的呈現可能會影響學習效率。

2. 研究目的

我們的研究目的就是要設計一語法搜尋機制，讓使用者可以輸入特定語法以查詢包含此語法的例句或文件。此語法搜尋機制應包含下列特性：

快速：雖然語法搜尋較關鍵字查詢複雜，但一個回應速度慢的搜尋機制會讓使用者望之卻步，所以設計一快速的搜尋機制是我們的首要目標。

彈性：我們設計的方法應可套用在任意的語料上，不只是標準語料庫，舉凡可成為學習目標的語言材料都可套用我們的搜尋機制，例如電影的字幕，歌詞等等。

使用方便：我們將提供數種查詢語法，讓使用者能輕易的設計出想要查詢的語法。

除此之外，我們也將設計一完整系統充分運用此搜尋機制，由於利用電影學英文已是一種趨勢，所以我們將此搜尋機制套用在電影字幕資料庫上，並將此系統套用在 IWiLL 語言學習平台的教材編輯工具(Authoring tool)上。透過這樣的結合，充分展現出以上所提的三個特點，以及其在英文學習上的作用。

3. 文獻探討

就我們所知，很少有研究設計語法搜尋，唯一一篇與語法搜尋相關的論文是由 Corley 教授等所提的 Finding Syntactic Structure in Unparsed Corpora[6]，其內容是利用語法剖析器(Parser)，將句子的語法剖析樹(Parsing Tree)建出來，再利用語法子樹對完整的剖析樹作搜尋比對。雖然此研究的方法新穎且提供了一個可行的語法搜尋機制，但卻不符合我們在研究方法中所提及的快速及使用方便性的特點。由於此方法需利用樹與樹之間的比對，勢必對效率造成影響，另外也不是任何使用者都有能力建立出一個查詢子樹，所以此法對我們所要設計的搜尋機制貢獻不大。

在我們的設計方法中，常規表示式(Regular Expression)占了很重要的角

色。我們利用常規表示式及語言中的詞性標記(Part-of-speech Tag)組合成語法查詢語(Syntax Query)，再利用已索引過的語料庫進行查詢比對。所以一個適合常規表示式的搜尋機制將會使我們的搜尋速度大為提升，我們採用的索引方法主要是參考 Chor 教授及 Rajagopalan 教授的論文：A Fast Regular Expression Indexing Engine[5]再加以改進成適合語法搜尋的機制，我們會在後面的章節詳述。

4. 研究方法

4.1. 前處理

本研究中使用以下的步驟處理各式語料，以使其適合進行語法搜尋：

1. 詞性標註 (Part-of-speech tagginig)
2. XML 格式化
3. k-gram 索引建立

以下詳述各個步驟：

1. 詞性標註 (Part-of-speech tagginig)

我們熟知的語法中皆由字彙及詞性構成，而原始的語料除了字彙並無詞性資訊，所以詞性標註對語法搜尋甚為重要。我們的詞性標註方式主要是參考 Brants 教授的論文： TnT -- A Statistical Part-of-Speech Tagger[4]，使用 BNC 作為訓練資料，經過內部測試可達到 96%的正確率。

2. XML 格式化

使用 XML 格式化的原因有二，第一是 XML 剖析器非常普遍，使語料易於交換；第二，XML 的屬性(Attribute)設計有利於語法查詢語的設計。

3. k-gram 索引建立

由於常規表示式是我們主要使用的查詢語，而常規表示式中最重要的一個特性就是萬用字元(wildcard)，為了實現包含萬用字元的查詢，我們將語料建立一 kgram 索引，下表為幾個索引範例：

Index term	所包含 index term 的字串
rel	rely、relax、relief、sorrel
wor	word、work、worst、worry
ion	action、union、ions、vision
ease	release、please、ease、cease
met	helmet、comet、method、metal

所以當查詢 rel*時，同時會查詢 rely、relax、relief 等。而完整的字彙則使用 Inverted list 的索引方式以加快搜尋速度。

4.2. 語料及查詢格式

我們以一簡單範例呈現語料經過前處理後的格式。

原始格式：I have to keep working.

<s>

<w Lemma="I" POS="PRN"> I </w>

<w Lemma="have" POS="VHB">have</w>

<w Lemma="to" POS="TO0">to</w>

<w Lemma="keep" POS="VVB">keep</w>

<w Lemma="work" POS="VVG">working</w>

</s>

而之前所提的語法查詢 keep + 現在進行式可用以下的常規表示式表示：

<[^>]+Lemma=\"keep\">[^>]+</w>\s*<[^>]*\SPOS=\"VVG\"[^<]*>[^>]+</w>

當然此種查詢一般使用者無法建立，所以我們也設計了一個建立語法查詢的介面，此介面將在實例解析中呈現。

更詳細的研究步驟請參考參考文獻[1]及附錄一、二。

4.3. 實例解析

我們將此搜尋機制套用在電影字幕資料庫上，並將此系統套用在 IWiLL 語言學習平台的教材編輯工具(Authoring tool)上。以下為實際操作步驟：

4.3.1 搜尋介面

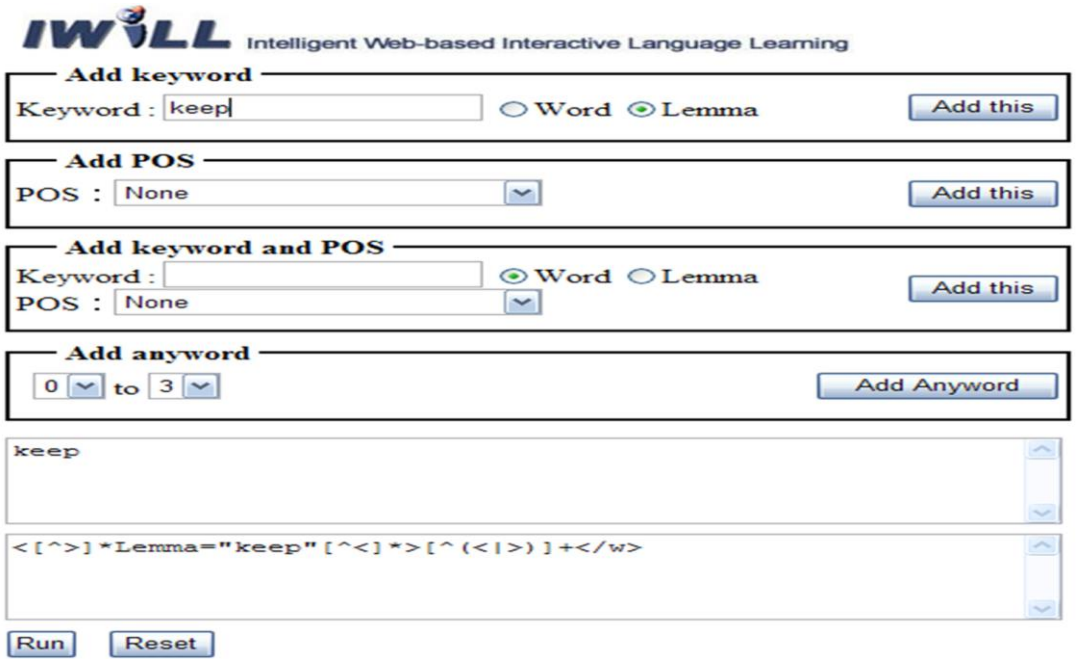
Add Keyword : 可以輸入你要查詢的單字，另有 Word 及 Lemma 可以選擇。

Add Pos : 選擇需要的詞性。

Add KeyWord and POS : 輸入要查詢單字及其詞性。

Add anyword : 輸入顯示長度。

當所有條件加入完成後，按下 Run 即可進行搜尋。



圖一

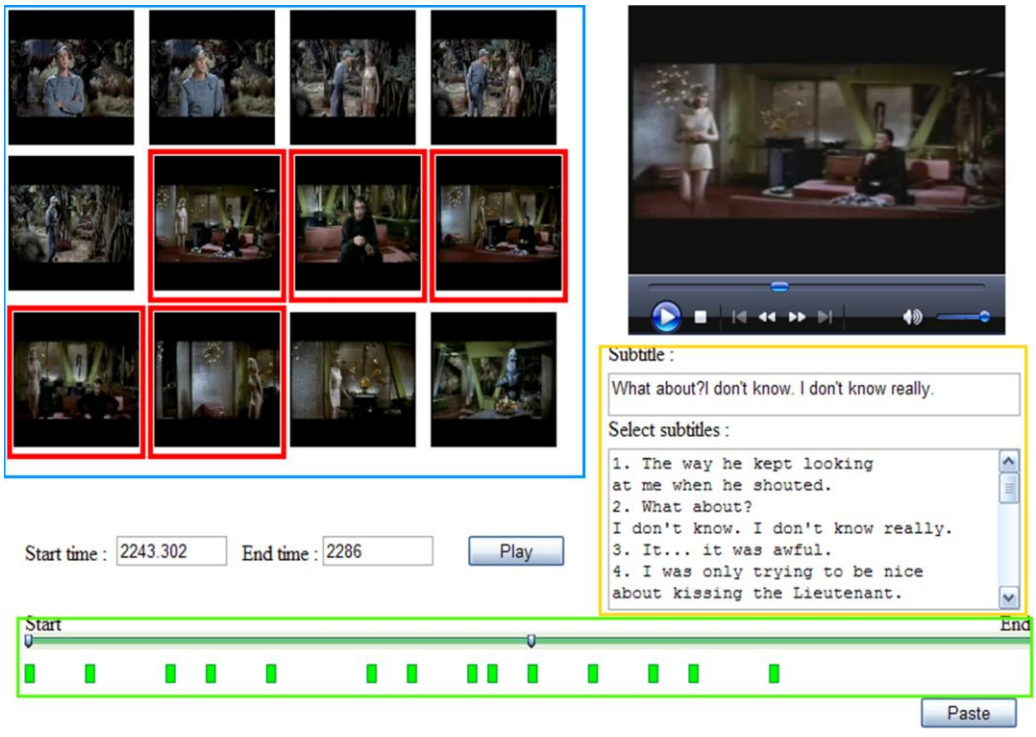
4.3.2 搜尋結果

左邊為搜尋結果，點選其中一個句子，右邊就可以撥放該句的影片片段。

圖二

4.3.3 教材及電影片段

左上藍色框內為該句子前後相關場景，下面綠色框內為選取場景中所有句子的分布狀況，右邊黃色框為字幕呈現區。



圖三

5. 參考文獻

[1]. Nai-Lung Tsao, Chin-Hwa Kuo, David Wible and Tsung-Fu Hung (2009) "Designing a Syntax-Based Retrieval System for Supporting Language Learning," Journal of Educational Technology & Society, Vol. 12(1), pp. 73–81.Chin-Hwa Kuo, David Wible, Nai-Lung Tsao, and Chen-Fu Chang, “A Video Retrieval System for Computer Assisted Language Learning,” AI-ED 2005, July 18-22, 2005.

[2]. Ruey-Jinng Chen, Chin-HwaKuo, Nai-LungTsao,and Tsung-Fu Hung CANLab.,Dept.ofCSIE,TamkangUniversity,Taiwan “Improving the Syntax-based Retrieval System Using Collocation Indexing” ICS2009 March 1, 2009.

[3]. 陳瑞璟 應用搭配詞概念提升語法搜尋系統之效能 淡江大學資訊工程學碩士論文 2008

[4]. Brants, T. (2000). TnT -- A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Applied Natrual Language Processing Conference ANLP-2000* (pp. 224–231). Seattle, WA.

[5]. Cho, J. & Rajagopalan, S. (2002). A Fast Regular Expression Indexing Engine. In *Proceedings of 18th IEEE Conference on Data Engineering* (pp. 419–430). San Jose, CA.

[6]. Corley, S., Corley, M., Keller, F., Crocker, M. W. & Trewin, S. (2001). Finding Syntactic Structure in Unparsed Corpora. *Computers and the Humanities, 35*, 81–94.

6. 計畫成果自評

計畫執行至今，在團隊的合作下，許多困難透過定期的團隊研討，亦已一一克服，上述的工具才得完成。此成果並已被 Educational Technology& Society (SSCI)期刊所接受(附錄一 Designing a Syntax-Based Retrieval System for Supporting Language Learning)。但此研究無論在搜尋速度及語法查詢的彈性上皆還有進步空間，所以希望國科會繼續支持後續之研究。

附錄一

Designing a Syntax-Based Retrieval System for Supporting Language Learning

Nai-Lung Tsao1, Chin-Hwa Kuo2, David Wible[1] and Tsung-Fu Hung[2]

[1]Graduate Institute of Learning and Instruction, National Central University, Taiwan, No.300, Jung-da Rd. JhongLi, Taiwan // Tel: +886-3-4227151-33851 // Fax: 886-3-4273371

[2]Department of Computer Science and Information Engineering, Tamkang University, Taiwan, No.151 Ying-chuan Road, Tamsui, Taiwan // Tel : +886-2-26215656-3304
beaktsao@gmail.com // chkuo@mail.tku.edu.tw // wible45@yahoo.com // kidd@mail.iwillnow.org

ABSTRACT

In this paper, we propose a syntax-based text retrieval system for on-line language learning and use a fast regular expression search engine as its main component. Regular expression searches provide more scalable querying and search results than keyword-based searches. However, without a well-designed index scheme, the execution time of regular expression search would be unacceptable to users. Our methods are based on Cho and Rajagopalan (2002) and we introduce some modifications, such as a presuf index constructing algorithm and a method for deciding minimum filter factor, to meet the requirements of our syntax-based text retrieval system. The experiment results show the index space size is small and the performance of syntax-based sample queries show significant improvements over benchmark results. A user-friendly query generator is designed to support users who have no background knowledge of regular expressions.

Keywords

Regular expression, Syntax-based retrieval, Indexing techniques, Query processing, Language learning

Introduction

The need for a well-design search engine is dramatically increasing because of the growing amount of data in real world, such as that on web pages, in standard text corpora and in movie databases. In addition to effective searching from massive amounts of data, recent search engines feature a flexible query language providing a wider variety of targeted search items. Compared to traditional text retrieval systems using keywords as basic query symbols, a system which provides regular expression as one of query languages seems more suited to meeting this requirement.

The main purpose of the syntax-based text retrieval system is to support grammatical querying of tagged corpora for language learners and teachers. While syntax-based queries contribute improvements to general-purpose queries of massive amounts of data, the power of regular expressions provide even further advantages when the users are language learners or teachers and their purpose is finding examples of specific types of language message. Consider the following example.

Example 1: In ESL teaching, it is important to teach learners the particular forms the certain verbs require of their complements, for example, keep requires a following verb in its –ing form (keep trying but not keep to try). If we want to find the form in any text corpora, traditional text retrieval systems have no good solutions but might scan all documents which include the keyword keep in the collection. Variant searching results are found, such as keep in touch and noun phrases followed by keep. That is because of the approaches to index construction, usually using inverted indices as index structure, in

traditional text retrieval systems are extracting useful keywords first and using these keywords as indices. A regular expression can easily present such a pattern. The regex (regular expression) below presents one possible way to describe this pattern.

keep\s+ \w+ing

Even though regular expressions provide more flexible querying, they still create a serious problem in terms of search response time. For example, a text collection with 1 million documents and 1,000 words average length would take an unacceptable response time, say, a couple hours, by match the above sample query pattern to strings of text. Without further processing, the only way to find the pattern is scanning each document one by one in the text collection.

There have been several proposals made to solve this search time problem. Most of them use k-gram index construction and build efficient index structures for quick searching of index terms. The index terms extracted from a text collection would be every sequence of characters of length k in each data unit. Once the index terms are extracted, the systems can construct suffix trees by using the technique described by Baeza-Yates and Gonnet (1996) or inverted indices in (Baeza-Yates and Navarro, 2004) (the most commonly used as index structure for k-gram indexing) to identify the data unit positions of each index term. By using the index, the search engine can only scan the documents or any data units which contain the specific targeted string in the regular expression query. For example, if a system performs k-gram index extraction from k=3-10, then the query in Example 1 can be matched just in data unit positions of index terms "keep" and "ing" rather than in the whole text collection. This simple idea can substantially reduce the search time. Every system would decide different ranges of k for specific purposes or considerations, such as limited secondary memory size. In this paper, the main approaches to building regex search engine are based on (Cho and Rajagopalan, 2002). The approaches use minimum index storage space and provide short enough search response time for most regex queries.

Example1 can be implemented in any regex enabled search engine. What if, however, we want to find the syntax without any specific string, such as *ing*? This would be valuable for learners who do not yet know what form requirements a particular verb places on its complement and want to discover this through examples. Example 2 shows another syntax pattern commonly targeted in ESL teaching.

Example 2: Collocation is a persistent area of difficulty for ESL learners. For example, what verbs can be used before the noun *problem*? If the corpus is part-of-speech tagged, we can use the following regex to search.

<w POS=\"V\W+\">\w+<\/w> (<w[^>]+><\/w>){0,5} <w[^>]+>problem</w>

where POS=\"V\W+\" indicates words tagged to *verb*.

Of course the text collection needs further processing to make the query work and we will discuss this below. However, we can see after adding some word class information to each word, such as part-of speech, the power of regex query can be greatly expanded by indicating specific strings of words or word classes or some combination of the two.

There are some different aspects of designing syntax-based text retrieval system from regex search engine. Cho and Rajagopalan (2002) provide a hardware I/O consideration for deciding the filtering factor, say, 0.1, while they assume the speed of sequential I/O access is ten times faster than random access. In this paper, we propose a different way to decide this factor concerning syntax-based text retrieval. The details of the approach are described in "Syntax-based text retrieval system" section.

The rest of the paper is organized as follows: in Section “Regular expression search engine”, the main approaches provided by Cho and Rajagopalan (2002) are described. The design details of syntax-based retrieval system are proposed in following section, including a user-friendly query generator. The conclusion and future works are presented in the last section.

Regular expression search engine

Figure 1 shows the main components of a typical regular expression search engine. The parts most different from traditional keyword-based search engines are the index construction and query processing components. The following subsections focus on these two components, which are proposed by Cho and Rajagopalan (2002).

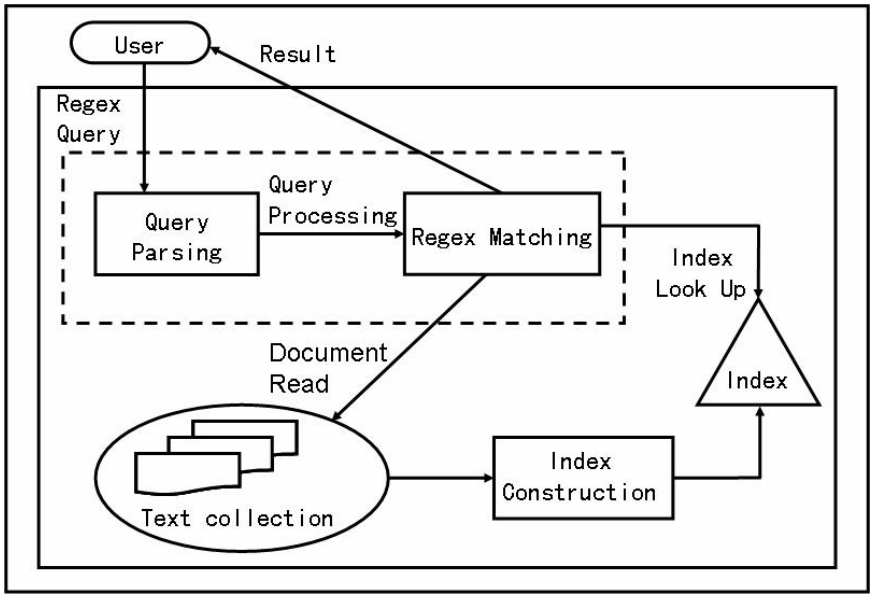


Figure 1: The illustrative figure of a typical regular expression search engine

Index construction

This subsection deals with the index construction algorithm and structure. First we introduce some notations and definitions. A k-gram is a string $x = x_1 x_2 \cdots x_k$, where $x_i : 1 \le i \le k$ is a character. A data unit means the unit in which the raw data is partitioned, such as web page, a paragraph or a sentence in documents. Let M(x) denote the number of data units which contain x and then the filter factor is denoted by $ff(x) = 1 - M(x)/N$, where N means the total number of data units. Now we can set the minimum filter factor *minff* and only keep index terms with filter factors greater than *minff*. This would make the number of index terms smaller and more useful. For example, *minff* = 0.95 means the system only keeps the index term which can filter 95% of data units. This filter scheme can make the index terms more discriminative, which is important for a retrieval system. We call these index terms **useful** indices.

Even if the system only maintains useful indices, the number of indices is still large. Every string expanded from a useful index will be useful, too. For example, if the index *NBA* is useful with in the text “*How to buy NBA tickets*”, then “*y NBA*” and “*NBA t*” are useful and it seems not necessary. Therefore, the system only maintains a *presuf* (prefix and suffix) free index set. A *presuf* free set means

there is no *x* in the index set is a prefix or suffix of any other index *x'*. For example, {ab, ac, abc} and {bc, ac, abc} are not presuf free because ab is a prefix of abc in first set and bc is a suffix of abc in second set.

After determining index terms, we can construct the index for a regex search engine. We use inverted indices as our index storage structure, which is easily accessed by RDBMS. The algorithm is shown as follow:

Input：text collection

Output：index

[1] k=1，Useless={.} // . is a zero-length string
[2] while (Useless is not empty)
[3] k-grams：= all k-grams in text collection
whose (k-1)-prefix∈Useless or (k-1)-suffix∈Useless
[4] Useless：={}
[5] For each x in k-grams
[6] If ff(x) ≥ *minff* Then
[7] insert(x,index) //the gram is useful
[8] Else
[9] Useless：=Useless∪{x}
[10] k：=k+1

In (Cho and Rajagopalan, 2002), the prefix-free indices are built first and then the suffix-free indices are considered. We reduce the number of passes needed for index construction from two passes to one pass and get speed improvement. The different part of our steps from the original one in (Cho and Rajagopalan, 2002) is we do prefix checking and suffix checking in the same pass. We need to scan entire data once for extracting all k-grams. After the index is built, it still needs one more entire data scanning for identifying index term positions. Therefore, the whole index construction needs only two entire data scannings without any extra memory space. However, if the memory is large enough, we can extract k-grams and get the positions in the same data scanning pass.

Query processing

As mentioned before, the k-grams index is used to reduce the number of data units to be matched by regex. For this reason, the query processor has to determine which index term to look up. The following shows the algorithm for determining candidate data units for regex matching.

Input：a regular expression query, *r*

Output：candidate data units

[1] Rewrite regular expression *r* so that it only uses $, (,), |
and &, where $ means entire data untis and & means
AND operator. The details are as follow:
1. Based on the order in Table 1, perform the action of

target symbol in *r*.

2. If there is no operator in any two operands, insert &

[2] Transform the infix expression to postfix expression

[3] Get k-gram which has the smallest position set contained in operand string.

[4] Perform the set operation among the position sets extracted in Step 3 and then get the final candidate data units for regex string matching.

In Step 1, the query processor refers to Table 1 to eliminate useless symbols for identifying index terms and replace the useless symbols with operators or specific operands. For example, because any string between [and] presents only one character and almost every unigrams are not discriminative, the query processor will eliminate every [] and the string between [and], and use a & mark to replace AND operator. For example, after being processed by Step 1, the regex in Example 2 will become *<w & POS=& >& <w &> (<w &> & <w &>) <w & >problem< & w>*.

In Step 2, the query processor transforms the infix expressions to postfix expressions thereby eliminating the need for operator and bracketing priorities. For example, A&(B|C) becomes ABC|&

Because not every k-gram is in index, the string operands in final postfix expression would not be available in index. Therefore, a strategy has to be provided to identify which index term to look up. We use a simple strategy: get k-gram which has the smallest position set. For example, an operand is *Compaq* and only two indexed k-grams contained in *Compaq* are *comp* and *paq*. The number of position set of *comp* is larger than the number of position set of *paq*. On this account, the query processor picks the position set of *paq* for generating candidate data units for regex string matching. This method is Step 3 in the algorithm.

Another thing that should be noticed is what happens if an operand contains no index terms in the index. Because of the useful index design, we cannot treat this situation as empty candidate set. Unfortunately, this operand must be treated as if all data units might contain this string. Based on this problem mentioned above, another problem raised here is after the position set operation, the candidate data units might be the whole data units. Two kinds of strategies might be used: (1) recommend users to change the query and (2) only list first n results for users. We will adopt the second method in our system.

Table 1: Regular expression symbols and handled actions

	Description	Action	Order
\	Escape character((\d) is any numeric characters)	Delete this symbol and the next character	1
[]	Any character in the bracket([a|b|c]=a|b|c)	Delete characters from"[" to"]"	2
+	One or more repetition of previous character	Delete this symbol	3
•	Any character(a.d match asd)	Delete this symbol	4
*	Zero or more repetition of previous character	Delete this symbol and replace with $	5
^	Any character except the next character	Delete this symbol and next token if next token is not blank	6

Syntax-based text retrieval system

As far as we know, there have been few researches about syntax-based search. Corley et al. (2001) proposed a syntactic structure search engine. Their system selects sentences by some grammar rules. This is achieved by a fast chart parser. Compared to their approach, our system needs no grammar rules

or parsers, but linear syntax knowledge. Our query language is simple and intuitive.

In this section we will discuss the design of syntax-based text retrieval systems. The index construction and query processing are just like the descriptions in previous section. We will focus on corpus design and evaluating sample queries.

Tagged corpus

As mentioned above, this syntax-based retrieval system is designed for language learning. We want to provide a system which can search grammatical information in a target text collection. Therefore the raw text has to be preprocessed for advanced searching. According to the grammar tutoring samples from high school textbooks, we decide to add part-of-speech information to the raw text. We train a Markov Model-based POS tagger (2000) and use British National Corpus (BNC, http://www.natcorp.ox.ac.uk/) as our training data. The internal evaluation shows this tagger has 93% precision including identifying unknown words.

The information of the tagged corpus also contains lemma of each word. The lemmatizing operation should be performed after POS tagging because some words have different lemma with different POS. The following shows an example after POS tagging and XML structure formatting.

```
<s>
    <w Lemma="do" POS="VDD"> Did </w>
    <w Lemma="you" POS="PNP">you</w>
    <w Lemma="know" POS="VVI">know</w>
    <w Lemma="?" POS="PUN">?</w>
</s>
```

We use XML to wrap the tagged text for three reasons. First, XML is suitable for representing grammatical information. Basically, XML and English can all be treated as a tree by visualization (Allen, 1995). Although our corpus only has POS information now, we think other grammatical information, such as phrase structure and relative clause identification, can easily be represented by XML. Second, many XML parsers have been produced. Using these XML parsers can easily extract information encoded by XML node inside and outside. Lastly, XML has been standard for data exchange. This will prepare our data to run on other applications some day.

Filter factor

We have discussed how Cho and Rajagopalan(2002) decide their minimum filter factor. This idea is not suitable for our system. Since our target users are English teachers and learners, they have more need of English vocabulary searching capacity but not special formats, such as URL or telephone numbers. Under this assumption, the index terms in our system should be token-oriented. Further, because length of most of English words is more than 3 and those words of length 1 or 2, such as *I, is,* and *to,* lack discriminative value for retrieval, we set our k from 3 to 10 (From the result of extracting experimental corpus, we found out there are about 3500 occurrences for 11-grams, which we think it's not worth to save the index.). As for the length 1 and 2 words, the system build the isolated keyword index for them. While meeting length 1 and 2 words in query processing, the process will search this isolated index. Furthermore, because we want all words of length 3 kept in the index set (to make the searching results nonempty), the index construction starts on k=4. We also think minff should be increased as k is increased because it's obvious that when the index length grows up, it can filter out more data.

Therefore we change the function to $minff(k) = 1 - \alpha^{k-k_0} \times \max M(x)/N$, where k_0 means the smallest number in k, in our case, $k_0 = 4$. α is a real number smaller than 1. α can be $1/52$ if no matter which letter of the English alphabet is added in front of or in back of the string with length k-1, each of them has the same $M(x)$.

Evaluation

We collect some sample syntax-based regex query results to evaluate the system performance. There are numerous queries which are interesting for high school teachers or relevant to textbook content. The following shows ten query samples.

1. Adj+respect
   <[^>]*\SOS=\"AJ\S{0,1}\"[^>]*>[^<]+</w>\s*<[^>]*\SOS=\"N\S{0,2}\"[^<]*>respect</w>

2. Verb+pain
   <[^>]*\SOS=\"V\S{0,2}\"[^<]*>[^>]+</w>\s*(<[^>)]+>[^>]+</w>\s*){0,4}<[^>]*\SOS=\"N\S{0,2}\"[^<]*>pain</w>

3. break +PREP
   <[^>]*POS=\"V\S{0,2}\"[^<]*>(break\S|broke)</w>\s*<[^>]*POS=\"PR\S{0,1}\"[^<]*>[^>]+</w>

4. of + WH-Word
   <[^|>]+>of</w>\s*<[^>]*\SOS=\"(PNQ|DTQ)\"[^<]*>[^>]+</w>

5. approach + to + Ving
   <[^>]+>approach\w{0,2}</w>\s*<[^>]+>to</w>\s*<[^>]+>\w+ing</w>

6. on the other hand
   <[^>]+>On</w>\s*<[^>]+>the</w>\s*<[^>]+>other</w>\s*<[^>]+>hand</w>

7. no matter
   <[^>]+>no</w>\s*<[^>]+>matter</w>

8. as+Adj+as
   <[^>]+>as</w>\s*<[^>]*\SOS=\"AJ\S{0,1}\"[^<]*>[^>]+</w>\s*<[^>]+>as</w>

9. worth+Ving
   <[^>]+>worth</w>\s*<[^>]*\SOS=\"VVG\"[^<]*>[^>]+</w>

10. to+Ving
    <[^>]+>to</w>\s*<[^>]*\SOS=\"VVG\"[^<]*>[^>]+</w>

The testing database is a subcorpus from BNC. We collect about 1,750,000 sentences, about 22,000,000 words, from BNC. All sentences, or say, data units, are POS tagged, lemmatized and XML formatted. The index construction approaches we use differ in some respects from (Cho and Rajagopalan, 2002). We do not extract k-grams from a sentence, but from keywords because we think the phrase searching can also be achieved by this method and for the above reason, this method can reduce the index storage. Table 2 shows the index sizes and posting sizes of complete index construction and presuf free index construction.

Table 3 shows the number of each candidate and final result number of each query and Figure 2 shows the execution time of the ten query samples by entire data search and our regex search engine. Most of the queries show a remarkable improvement by using our regex search engine. Those which show no

improvement are due to the regex strings containing no useful index term. These unsuccessful queries contain no substring of keywords in sentences but POS tags strings. Even if the system treats this POS tags as index terms, these index terms almost all refer to entire data units or half of them. We treat this problem as one of our future works.

Table 2: Sizes of index terms and postings

	Complete k-grams index	Presuf-free set
Number of Index Terms	968,745	10,339
Numbers of Postings	166,952,527	53,754,273

Table 3: Numbers of candidate set and final results. Total number: 1,777,516

Query	Candidate	Final
1	27,421	215
2	18,232	929
3	4,588	156
4	1,777,516	10250
5	107,516	6370
6	82,266	630
7	47,740	29
8	1,777,516	5070
9	35,101	870
10	1,777,516	4300

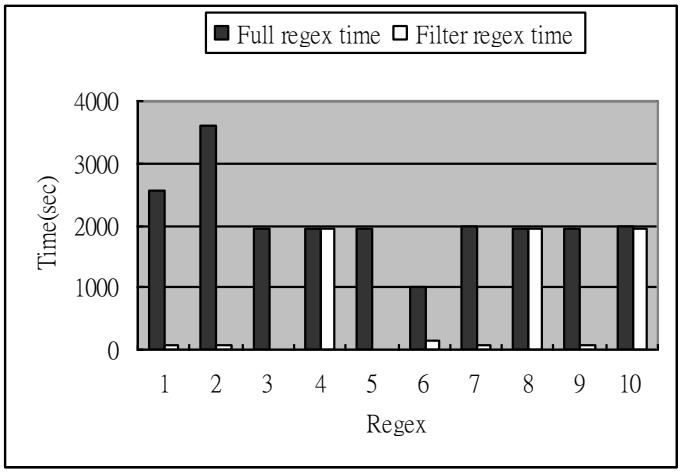


Figure 2: Execution time of each query

Query Generator

Although the syntax-based retrieval system yields impressive performance gains, it is not easy to use for our target users, English teachers and learners. For this reason, we also have designed a query generator for those users who have no knowledge of regular expressions. The query generator is shown in Figure 3. Since our query language is linear, users construct a query element by element. Here an element means one grammatical unit, such as a word, a lemma, or a word class.

As shown in Figure 3, the user interface consists of three parts, namely, (1) keyword, (2) POS, and (3) anyword to construct the desired syntax form. The resulting format is shown in query input box (see legend 4 in Figure 3.) Note that the corresponding regex form can appear as users press the "Add this" button. Also note that the operation sequence of the above parts depends on the desired syntax format. In other words, it does not necessarily follow the order shown in Figure 3.

We make use of the following examples to illustrate the above description.

Example 3, single word: To retrieve sentences contain the keyword *wonderful*, user simply type in *wonderful* in the keyword part, and press add button. Then the query input is appeared in query input box.

Example 4, syntax format: To retrieve the syntax format:

"verb *keep* followed by -ing form verb"

First we type in *keep* as first query term (legend 1 in Figure 3). Since the -ing word may be allocated at several words right from the word *keep*, we can insert words for this elasticity of demand (legend 1 in Figure 3). In this case, we add 0 to 5 "Anyword" patterns, which means –ing word is far away from *keep* at most 5 words. The last query term, of course, is the part-of-speech "-ing form verb" (legend 2 in Figure 3). The corresponding syntax query is generated (keep + <anyword>(0,5) + Ving) as legend 4 in Figure 3.

Example 5, syntax format: To retrieve the syntax format:

"adjective before the word *rain*"

First, we choose the adj term from POS term. Then, add any other words (e.g., from zero to three words). Finally, we type in the word rain in the keyword term. The corresponding syntax query is generated (adj + <anyword>(0,3) + rain).

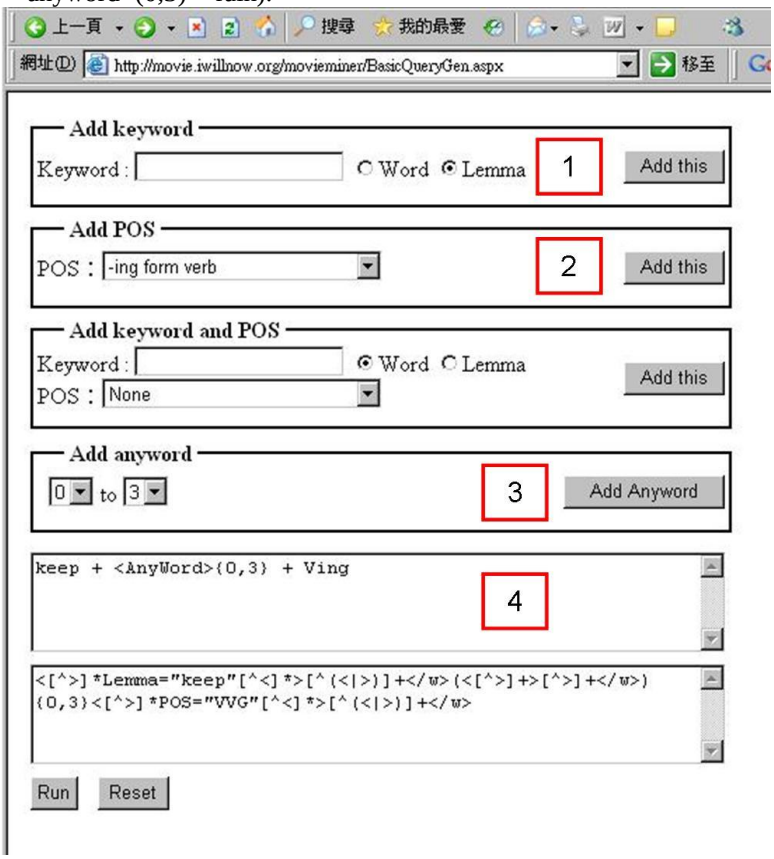


Figure 3: The designed User Interface of Query Generator

Conclusion

We have described how to design a syntax-based text retrieval system for language learning. The system introduces a regular expression search engine to reduce response time. Grammatical annotation of the target texts helps the system to provide syntax-based searches to locate tokens of specific types of target language use, thus providing valuable supports for language teaching and learning. However, there is still no solution for making class-based queries faster because they have no index available to reduce the search time. Enrichment of the corpus information, for example with semantic tagging, is also one important work for future research.

References

Allen, J. (1995). *Natural language understanding*, CA: Benjamin/Cummings Publishing Company.

Baeza-Yates, R. & Gonnet, G. H. (1996). Fast Text Searching for Regular Expressions or Automation Searching on Tries. *Journal of the ACM,* 43, 6 (Nov), 915–936.

Baeza-Yates, R. & Navarro, G. (2004). Text Searching: Theory and Practice. In C. Martin-Vide, V. Mitrana & G. Paun (Eds.) *Formal languages and applications* (pp.565–597). Berlin: Springer.

Brants, T. (2000). TnT -- A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Applied Natrual Language Processing Conference ANLP-2000* (pp. 224–231). Seattle, WA.

Cho, J. & Rajagopalan, S. (2002). A Fast Regular Expression Indexing Engine. In *Proceedings of 18th IEEE Conference on Data Engineering* (pp. 419–430). San Jose, CA.

Corley, S., Corley, M., Keller, F., Crocker, M. W. & Trewin, S. (2001). Finding Syntactic Structure in Unparsed Corpora. *Computers and the Humanities, 35,* 81–94.

附錄二

Improving the Syntax-base Retrieval System Using Collocation Indexing

Ruey-Jinng Chen, Chin-Hwa Kuo, Nai-Lung Tsao, and Chen-Fu Chang
CAN Lab., Dept. of CSIE,
Tamkang University, Taiwan,
695410596@s95.tku.edu.tw, *chkuo@mail.tku.edu.tw*, *beaktsao@mail2000.com.tw*,
kidd@mail.iwillnow.org

Abstract

The purpose of this paper is to design a syntax search system and to apply it to a movie search system. The concepts applied include those in the field of linguistics and collocation, to increase the speed of the syntax search system. First, we must process the keywords in the database by labeling them according to their part of speech. From the results of the process, we will construct a K-gram index and Collocation index. In this proposal we bring out a few examples of common English syntax rules and sentence structures as test models. After the run through, the K-gram index and the Collocation index are compared. We have found that part of the sentence, after having gone through the Collocation index search, has a far smaller sample space that the K-gram index alone, which is to say that the Collocation index is able to find the most correct result from fewer samples, thus minimizing the time cost in Query Match.

Kerywords：POS tagging, Lemmatizing, Collocation, k-gram, Indexing

1. Introduction

Due to the swift expansion of the Internet, whereas words and text were the main medium of transmitting information, they are replaced with multimedia products. The combination of the multimedia and the Internet represents the future trend and model. In general, when talking about multimedia, people think about images, music, and visual aids, and entertainment. In recent years, research has shown that if multimedia can greatly aid education [1][2]. One of the liveliest discussions involves the combination of movies in the teaching of English and digital learning.

We have designed a system in order to provide teachers and students to search for syntax through movies dialogues. Like regular keyword search systems, we included POS tagging and lemmatizing to each of the keywords in the dialogues. This information is packaged into XML mode, and through the use of Regular expressions, we can easily obtain the information we need. Also, we used K-gram indexing [10][11]13] and Query match to obtain an basic syntax search result.

The most important step in a search process is obtaining the information. A significant topic of research targets how to increase the efficiency of the search process and to increase the accuracy of the results.

Therefore this proposal uses the concepts of language learning to increase the quality of these parts. Michael McCarthy [4] states that Collocation is a group of words that commonly appear together. Often, this group encounters many restrictions in terms of its application. Collocation needs to calculate the frequency

that two words appear together. Whether this type of combination is coincidental or not is determined by the analysis of the database. Most of the grouping of the Collocations is very meaningful. Therefore the degree of the word-to-word relationship is a significant step.

Gledhill [5] also points out three different viewpoints concerning the Collocation: (i) statistical: from a statistical perspective, Collocation have been shown to commonly appear in certain positions [6][7]. (ii) construction, which sees collocation either as a correlation between a lexeme and a lexical-grammatical pattern, or as a relation between a base and its collocative partners (iii) expression, a pragmatic view of collocation as a conventional unit of expression, regardless of form.

The Collocation concept led us to constructing a Collocation index [14]. Through the Collocation index system, we can effectively decrease the number compared in the Query match, which improves the efficiency.

2. System Overview

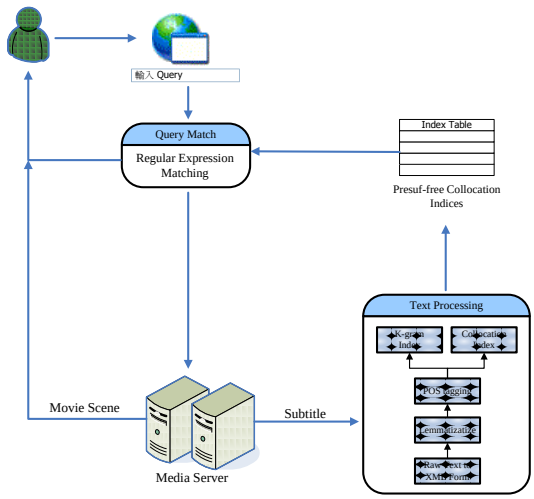


Figure 1. System diagram

The structure of the system is as shown in Figure 1. Media Serve is the system database, and contains the video clips and dialogue. After the dialogue has gone through text processing, it constructs a system query chart. The user can enter the keywords on the interface and the system will look for movie scenes and conversations.

2.1 Text Processing

In this system, text processing can be distinguished into three parts:

(1) Part of speech tagging - the purpose of part of speech tagging is to find the corresponding part of speech for the keyword. This way, the user may search for keywords according to the parts of speech.

(2) Lemmatizing - In this step, the word undergoes a lemmatizing process. For instance, a noun may be changed from plural to singular form, or present participle to present, as in "Paying" becomes "pay". Both part of speech tagging and lemmatizing offer an alternative way to examine the sentence structure.

(3) Change the original movie subtitles to XML form: Before a language search is performed, the movie subtitles must be converted to XML form. XML form includes the information obtained from POS tagging and Lemma and can be compared to Regular Expression.

2.2 Index Construction

An index is constructed for each keyword as the basis of searching, so that within a large amount of subtitles, one can quickly find the most suitable search. This system includes K-gram indexing and Collocation indexing, which will be described in greater detail in the next section.

3. Index Structure

In this part we will describe in detail how the K-gram index and Collocation index are constructed and the algorithms behind the index.

3.1 K-gram indexing

What K-gram means is that the K in the K-gram is

exchanged for different kinds of values, so that each keyword is divided into many grams. Then the index terms make up the search. The process is mainly composed of two steps, which are: (i) Multi-gram division (ii) index term search.

(i) A string of words *X*, of length n. A substring, where i is between 1 to n, length *k*, is the K-gram of the string.

(ii) The string is divided through Multi-gram, so there are different lengths of index terms. Therefore these new index term is the original String's index value, as shown in Table 1..

Table 1. Index of index term

Index	Index term's value
met	material、comet、helmet、method
tor	torrent、sector、torch、motor
wor	word、work、worry、worst
rel	release、rely、sorrel、relax
ease	release、please、ease、cease

3.2 Useful index

After the Multi-gram indexing is constructed, we will find many index terms in the Multi-gram index are not discriminative. Therefore, from the Useful index mentioned by Cho and Rajagopalan [10]'s research and Chen Fu Chang[11]'s research, we can decrease the number of index terms in the Multi-gram index.

Definition 3.1 *x* is any gram (index term) of M . M is a Data unit (word). There are N number of Data Units (word). M(*x*) is the Data Unit variable which includes gram *x*. Filter fact (*ff*) decides whether or not gram *x* should eliminate or retain a filter value. The algorithm is:

$$ff(x) = 1 - \frac{M(x)}{N}$$

From each gram x, we can calculate and select one. When it is greater, this means that the gram (index term) are useful, therefore it must be kept. From this concept, we can eliminate the gram (index terms) that do not matter and thereby reduce the number of index. The remaining gram (index term) are collected and become a "useful index".

Example 3.1 If we set $\min ff = 0.98$, it will filter 98% of data units, and only retain 2% of data units. These index terms are "useful."

3.3 Presuf free set

In order to retain the most useful index terms amidst a large number, we find that each index term that stems from a useful index term is also useful. Therefore we use the concept of presuf (prefix and suffix) free set to solve this problem.

Example 3.2 In the sentence "I am one of the CAN Lab member." CAN is useful, so any gram (index term) that comes from CAN is also useful, such as "e CAN" or "CAN La". Actually, there is no need; if we find this sentence by searching using the index term "CAN", we can also find the same sentence from searching "e Can" or "CAN La". However, these two index terms are unable to find more sentences containing "CAN".

Definition 3.2 Presuf free set: in this index set, an index term that does not exist will be the prefix or suffix of another index term.

3.4 Collocation indexing

In the introduction we often mention that collocation can be seen from a syntax viewpoint, a statistical viewpoint as well. From a syntax perspective, the decision is made from the part of speech. On the other hand, from a statistical perspective, the word-to-word relationship and position makes the

decision. The two following steps help use search of the collocations within the sentences: (i) Collocation construct (ii) Collocation filtering.

Collocation Construct：We focus on the Part of Speech tagging for each term. The basis of this process is from BNC (British National Corpus) [12]. The part of speech definitions are varied in the BNC, so we have simplified them into verbs, nouns, adjectives, adverbs, prepositions, conjunctions, and WH-word. We need to verify the positional relationship between the words. To do this, we created a standard where when the two words are part of the above categories and separated by no more than five words, we call it a Collocation.

Collocation Filtering：After Collocation is constructed through the syntax perspective, many Collocation index terms are produced, but not all of them are commonly used in English. First, for the positional relationship, we use a look-ahead method, that is, if the Pre_word_idx (the index of the preceding word) is greater than the Next_word_idx (the index of the following word), this index terms is eliminated. To determine how useful the collocation is, we still need to see how often it appears in the essay. If it only appears a few times in the entire database , this means that people do not normally use the words in this manner, so it is also eliminated.

Definition3.3 A collocation index term C_k, to be part of a perfect collocation set, must fulfill the following two requirements:

1. Pre_word_idx < Next_word_idx
2. $Freq(C_k) > T$

When the Collocation is greater than the filter value *T*, it becomes an index-term. These index terms are not only helpful to language learning, but more importantly, it can decrease the number of index terms that need to be compared in regular expression, therefore increasing the efficiency.

3.5 Term Search System

In order for the user to quickly search for syntax within a large collection of dialogue, we use the K-gram indexing mechanism, then decrease the index number through presuf set and useful index, as shown Figure 2. Then the Collocation index adds the filtered part of the presuf set and useful, so that the system's index becomes more complete.

From 3.1, 3.2, 3.3 we can obtain the following:

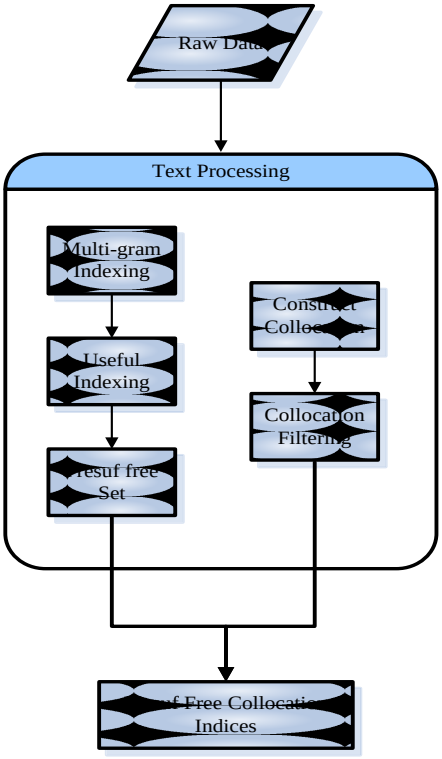


Figure 2. Architecture of subsystem

```
Input：text collection
Output：index

[1]  k=1，Useless={.} // . is a zero-length string
[2]  while (Useless is not empty)
[3]   k-grams：= all k-grams in text collection
                 whose (k-1)-prefix ∈ Useless
                    or (k-1)-suffix ∈ Useless
[4]   Useless：={}
[5]   For each x in k-grams
[6]        If ff(x) ≥ minff Then
[7]           insert(x,index) //the gram is useful
[8]        Else
[9]           Useless：=Useless ∪ {x}
[10]  k：=k+1
```

Figure 3. K-gram indexing algorithm

```
Input：Sentence collection
Output：index

[11]   Collocation = {.}
[12]   For each word in Sentence
                    //construct collocation
[13]  If (Pre_word_idx - Next_word_idx) < 1
[14]       (Pre_word + Next_word) ∈ Collocation
[15]   For each y in Collocation
[16]       If (Frequency(y) < min_threshold)
[17]            Delete y from Collocation
                 //the collocation y is not significant
```

Figure 4. Collocation indexing algorithm

4. Experiment

The syntax search system in this paper is a web-based system, which means the browser is used both to search and to display the search results. Figure 5 shows a screenshot of the interface:

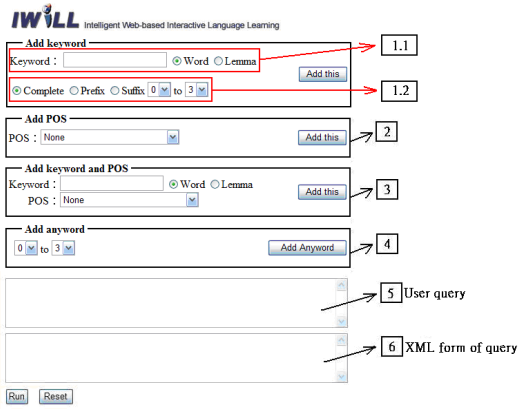


Figure 5. The designed user interface of query generator

From Figure 5, the Add Keyword portion can be divided into two parts. In Figure 5-(1.1), the user can enter a Keyword under "Keyword", and then select Word or Lemma method. If the user chooses word, the Prefix, Suffix, and Complete options are also available (Figure 5-(1.2)). If the user chooses prefix, the Suffix of the search results, and the two values can control the preceding character area of the Keyword. Lemma will return all the types of sentences from that word back to the user. After confirming the search requirements, the user clicks "Add this", and the search keyword will be converted to ser search syntax and regular expression, as shown in Figure 5-5 and Figure 5-6.

Add POS (Figure 5-2) is based primarily on Parts of Speech search. We provided 11 kinds of common parts of speech for the user's selection. In addition, "Add Keyword" is available. For example, keep + V-ing shows how these two go together.

Add Keyword and POS portions (Figure 5-3) , the Keyword typed by the user can not only select Word or Lemma, but also select the part of speech of the keyword.

Add Anyword (Figure 5-4)'s main function, is that when the user hopes to find how words work together, this collocation often has a great distance from word-to-word, rather than following one after another. For instance, "Add keyword" and "Add anyword" can be applied to the words "do… favor" to obtain this effect.

4.1 Experiment and Discussion

This paper mainly researches on the collocation indexing technology and its ability to improve the original system's functions. Also we increased the number of search results. To further this discussion, we used 119 movies, 161,900 sentences from dialogues as our training data.

Through our training data process, we use 15 sentences that are commonly used. Figure 6 shows the sentences that we tested in Regular Expression.

11. used to
    <[^(<|>)]+>used</w><[^(<|>)]+>to</w>
12. (could | would | should) + have + p.p
    <[^(<|>)]+>\w{1,2}ould</w><[^(<|>)]+>have</w><[^>]*POS="VVN"[^<]*>[^(<|>)]+</w>
13. approach(es | ing) + to + Ving
    <[^(<|>)]+>approach\w{0,2}</w><[^(<|>)]+>to</w><[^>]*POS="VVG"[^<]*>[^(<|>)]+</w>

1. of + WH-Word
<[^(<l>)]+>of</w><[^>]*POS="(AVQlDTQlPNQlAVQ-CJS)"[^<]*>[^(<l>)]+</w>
2. on + the + other + hand
<[^(<l>)]+>on</w><[^(<l>)]+>the</w><[^(<l>)]+>other</w><[^(<l>)]+>hand</w>
3. to + Ving
<[^>]*Lemma="to"[^<]*>[^(<l>)]+</w><[^>]*POS="VVG"[^<]*>[^(<l>)]+</w>
4. no + matter
<[^>]*Lemma="no"[^<]*>[^(<l>)]+</w><[^>]*Lemma="matter"[^<]*>[^(<l>)]+</w>
5. must + have + p.p
<[^(<l>)]+>on</w><[^(<l>)]+>the</w><[^(<l>)]+>other</w><[^(<l>)]+>hand</w>
6. so + Adj + that
<[^(<l>)]+>so</w><[^>]*POS="AJ0"[^<]*>[^(<l>)]+</w><[^(<l>)]+>that</w>
7. so + Adj + as to
<[^(<l>)]+>so</w><[^>]*POS="AJ0"[^<]*>[^(<l>)]+</w><[^(<l>)]+>as</w><[^(<l>)]+>to</w>
8. At + the + (sight l thought)+ of
<[^(<l>)]+>at</w><[^(<l>)]+>the</w><[^(<l>)]+>\w{2,4}ght</w><[^(<l>)]+>of</w>
9. as + <AnyWord>{0,3} + say*{0,4}
<[^(<l>)]+>as</w>(<[^>]+>[^>]+</w>){0,3}<[^(<l>)]+>say\w{0,4}</w>
10. Once + <AnyWord>{0,5} + -s form verb
<[^>]*Lemma="Once"[^<]*>[^(<l>)]+</w>(<[^>]+>[^>]+</w>){0,5}<[^>]*POS="VVZ"[^<]*>[^(<l>)]+</w>
14. if + only
<[^>]*Lemma="if"[^<]*>[^(<l>)]+</w><[^>]*Lemma="only"[^<]*>[^(<l>)]+</w>
*>[^(<l>)]+</w><[^(<l>)]+>as</w>
15. as + Adv + as
<[^(<l>)]+>as</w><[^>]*POS="AV0"[^<]*>[^(<l>)]+</w><[^(<l>)]+>as</w>

Figure 6. 15 sentences and its regular expression

Table 2 shows the results from our experiment shown in Figure 6. In particular, the Candidate column is added separately to the number of sentences before and after it is paired with the collocation index. Figure 7 is the statistical results of Table 2. The Final column represents the final number after Candidate and Query. From looking at Candidate (with collocation index) in Figure 7, it appears to be less than Candidate (without collocation index).

After adding the filtering collocation indexing, it can effectively decrease the number of times needed for comparison during regular expression, and can increase the speed for searching. This can improve the effectiveness of language search, which is the main contribution of this paper.

Table 4.1 The different from (with / without) Collocation index

Query	without collocation		with collocation	
	Candidate	Final	Candidate	Final
1	3410	120	90	120
2	21190	119	21190	119
3	520	0	520	0
4	161900	0	61480	35
5	8160	8	8160	8
6	161900	0	99360	18
7	10200	64	10200	64
8	8580	39	270	39
9	92450	9	14620	9
10	111420	0	920	3
11	58720	3	15480	3
12	27420	31	530	31
13	7030	2	1450	2
14	10960	6	100	6
15	14400	49	4270	49

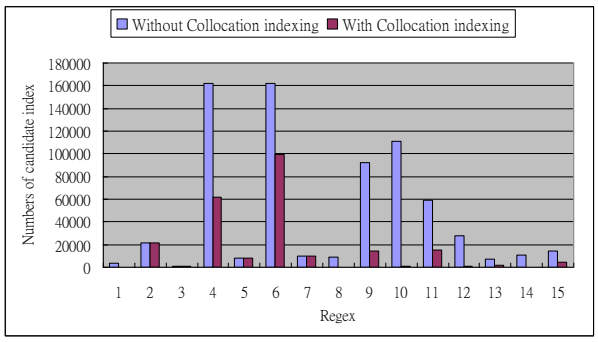


Figure 7. Total number of sentences between (with/without) Collocation index

5. Conclusion

This paper mainly explores whether or not the collocation indexing method can improve K-gram indexing search, and increase the number of search results. In the experiments, we find that the assistance of the collocation index enables sentences that could not originally be captured it to be easily found, as shown in Fig. 4.2. Therefore, collocation index's assistance greatly improves the retrieval system.

6. Reference

[1]. Jane King, "Using DVD Feature Films in the EFL Classroom," ,*Computer Assisted Language Learning*, Vol. 15, No. 5, pp 509-523, 2002.

[2]. Erwin Tschirner, "Language Acquisition in the Classroom: The Role of Digital Video," *Computer Assisted Language Learning*, Vol. 14, No. 3-4, pp 305-319, 2001.

[3]. Thorsten.Brants, TnT-A Statistical Part-of-Speech Tagger. In Proceedings of the Sixth Applied Natrual Language Processing *Conference ANLP-2000*, Seatle,WA, 2000.

[4]. McCarthy, Michael and O'Dell, Felicity, English Collocations in Use : how words work together for fluent and natural English. Cambridge University Press, 2005.

[5]. Gledhill C., Collocations in Science Writing, Narr, Tübingen, 2000.

[6]. Sinclair J. "The Search for Units of Meaning", in Textus, IX, 75-106, 1996

[7]. Smadja F. A & McKeown, K. R. "Automatically extracting and representing collocations for language generation", *Proceedings of ACL* 90, 252-259, Pittsburgh, Pennsylvania, 1990

[8]. Moon R. Fixed Expressions and Idioms, a Corpus-Based Approach. Oxford, Oxford University Press, 1998

[9]. Frath P. & Gledhill C. "Free-Range Clusters or Frozen Chunks? Reference as a Defining Criterion for Linguistic Units," In *Recherches anglaises et Nord-americaines*, vol.38：25-43, 2005

[10]. Cho, Junghoo and Sridhar Rajagopalan, "A Fast Regular Expression Indexing Engine." In *Proceedings of 18th IEEE Conference on Data Engineering*, 2002

[11]. Chin-Hwa Kuo, David Wible, Nai-Lung Tsao, and Chen-Fu Chang, "A Video Retrieval System for Computer Assisted Language Learning," *AI-ED 2005*, July 18-22, 2005.

[12]. BNC http://www.natcorp.ox.ac.uk/.

[13]. Ryoichi Ando, Koichi Shinoda, Sadaoki Furui, Takahiro Mochizuki "Robust Scene Recognition Using Language Models for Scene Contexts" In *Proceedings of the 8th ACM international workshop on Multimedia Information Retrieval*, 2006

[14]. Petrovic, S. Snajder, J. Dalbelo-Basic, B. Kolar, M. "Comparison of Collocation Extraction Measures for Document Indexing" *Information Technology Interfaces ITI 2006*, June 19-22, 2006