Language Combinatorics: A Sentence Pattern Extraction Architecture Based on Combinatorial Explosion

Michal Ptaszynski *ptaszynski@hgu.jp*
High-Tech Research Center
Hokkai-Gakuen University
Sapporo, 064-0926, Japan

Rafal Rzepka *kabura@media.eng.hokudai.ac.jp*
Graduate School of Information Science and Technology
Hokkaido University
Sapporo, 060-0814, Japan

Kenji Araki *araki@media.eng.hokudai.ac.jp*
Graduate School of Information Science and Technology
Hokkaido University
Sapporo, 060-0814, Japan

Yoshio Momouchi *momouchi@eli.hokkai-s-u.ac.jp*
Department of Electronics and Information Engineering,
Faculty of Engineering
Hokkai-Gakuen University
Sapporo, 064-0926, Japan

Abstract

A "sentence pattern" in modern Natural Language Processing is often considered as a subsequent string of words (n-grams). However, in many branches of linguistics, like Pragmatics or Corpus Linguistics, it has been noticed that simple n-gram patterns are not sufficient to reveal the whole sophistication of grammar patterns. We present a language independent architecture for extracting from sentences more sophisticated patterns than n-grams. In this architecture a "sentence pattern" is considered as n-element ordered combination of sentence elements. Experiments showed that the method extracts significantly more frequent patterns than the usual n-gram approach.

Keywords: Pattern Extraction, Corpus Pragmatics, Combinatorial Explosion.

1. INTRODUCTION

Automated text analysis and classification is a typical task in Natural Language Processing (NLP). Some of the approaches to text (or document) classification include Bag-of-Words (BOW) or n-gram. In the BOW model, a text or document is perceived as an unordered set of words. BOW thus disregards grammar and word order. An approach in which word order is retained is called the n-gram approach, proposed by Shannon over half a century ago [22]. This approach perceives a given sentence as a set of n-long ordered sub-sequences of words. This allows for matching the words while retaining the sentence word order. However, the n-gram approach allows only for a simple sequence matching, while disregarding the grammar structure of the sentence. Although instead of words one could represent a sentence in parts of speech (POS), or dependency structure, the n-gram approach still does not allow for extraction or matching of more sophisticated patterns than the subsequent strings of elements. An example of such pattern, more sophisticated than n-gram, is presented in top part of Figure 1. A sentence in Japanese "*Kyō wa nante kimochi ii hi nanda !*" (What a pleasant day it is today!) contains a syntactic pattern

"*nante * nanda !*"[1]. Similar cases can be easily found in other languages, for instance, English and Spanish. An exclamative sentence "Oh, she is so pretty, isn't she?", contains a syntactic pattern "Oh * is so * isn't *?". In Columbian Spanish, sentences "*¡Qué majo está carro!*" (What a nice car!) and "*¡Qué majo está chica!*" (What a nice girl!) contain a common pattern "*¡Qué majo está * !*" (What a nice * !). With another sentence, like "*¡Qué porquería de película!*" (What a crappy movie!) we can obtain a higher level generalization of this pattern, namely "*¡Qué * !*" (What a * !), which is a typical *wh*-exclamative sentence pattern [15]. The existence of such patterns in language is common and well recognized. However, it is not possible to discover such subtle patterns using only n-gram approach. Methods trying to go around this problem include a set of machine learning (ML) techniques, such as Neural Networks (NN) or Support Vector Machines (SVM). Machine learning has proved its usefulness for NLP in text classification within different domains [21, 16]. However, there are several problems with the ML approach. Firstly, since machine learning is a self-organizing method, it disregards any linguistic analysis of data, which often makes detailed error analysis difficult. Moreover, the statistical analysis performed within ML is still based on words (although represented as vectors), which hinders dealing with word inflection and more sophisticated patterns such as the ones mentioned above. Although there are attempts to deal with this problem, like the string kernel method [12], in ML one always needs to know the initial training set of features to feed the algorithm. Finally, methods for text classification are usually inapplicable in other tasks, such as language understanding and generation.

In our research we aimed to create an architecture capable to deal or help dealing with the above problems. The system presented in this paper, SPEC, extracts from sentences patterns more sophisticated than n-grams, while preserving the word order. SPEC can work with one or more corpora written in any language. The corpora can be raw or preprocessed (spaced, POS tagging, etc.). This way SPEC extracts all frequent meaningful linguistic patterns from unrestricted text. This paper presents general description of SPEC, evaluates several aspects of the system performance and discusses possible applications.

The paper outline is as follows. In section 2 we present background and motivation for the research, and explain general terms frequently used in this paper. Section 3 contains detailed description of all system procedures and modules put to the evaluation. In section 4 we describe the experiments performed to evaluate the system in several aspects influential for its performance. Finally, we discuss the results in section 5 and conclude the paper in section 6.

2. BACKGROUND

2.1 Corpus Pragmatics

Pragmatics is a subfield of linguistics focusing on the ways natural language is used in practice [11]. In general it studies how context of a sentence influences its meaning. There are, roughly, two approaches to this problem. Classic approach is to look for "hidden meanings" of a sentence, called implicatures [6]. Another approach takes as an object a corpus (a coherent collection of texts) and analyzes examples of certain language strategies within their contexts to study their functions. For this it has been named Corpus Pragmatics (differently to Corpus Linguistics, which does not put so much focus on context, but rather on the word examples alone). Some of the research in Corpus Pragmatics has been done by Knight and Adolphs [7], Potts and Schwarz [15], or Constant, Davis, Potts and Schwarz [4]. Especially the latter two have focused on emotive utterances. They have used a corpus of reviews from Amazon.com, and analyzed tokens (words, usually unigrams to trigrams) that were the most distinguishable for emotively emphasized reviews (marked very low or very high). The main drawback of these research was focusing only on words, while disregarding both grammatical information, like POS or dependency structure, and more sophisticated patterns, like the ones mentioned in Introduction. With this paper we wish to contribute to the field of Corpus Pragmatics by providing an architecture capable of automatic

[1] equivalent of *wh*-exclamatives in English [20]; asterisk used as a wildcard.

extraction of such patterns from corpora. In our assumption this could be done by generating all patterns as ordered combinations of sentence elements and verifying the occurrences of all patterns within a corpus. This introduces to our research a problem of Combinatorial Explosion.

2.2 Combinatorial Explosion

Algorithms using combinatorial approach generate a massive number of combinations - potential answers to a given problem. This is the reason they are sometimes called brute-force search algorithms. Brute-force approach often faces the problem of exponential and rapid grow of the function values during combinatorial manipulations. This phenomenon is known as combinatorial explosion [9]. Since this phenomenon often results in very long processing time, combinatorial approaches have been often disregarded. We assumed however, that combinatorial explosion can be dealt with on modern hardware to the extent needed in our research. Moreover, optimizing the combinatorial approach algorithm to the problem requirements should shorten the processing time making combinatorial explosion an advantage in the task of pattern extraction from sentences.

2.3 Pattern Extraction

Pattern extraction from language corpora is a subfield of Information Extraction (IE). There is a number of research dealing with this task or applying pattern extraction methods to solve other problems. Some of the research related the most to ours include Riloff 1996 [18], Uchino et al. 1996 [25], Talukdar et al. 2006 [24], Pantel and Pennacchiotti 2006 [14] or Guthrie et al. [23]. Riloff [18] proposed AutoSlog-TS system, which automatically generates extraction patterns from corpora. However, their system, being in fact an updated version of previous AutoSlog, was created using manually annotated corpus and a set of heuristic rules. Therefore the system as a whole was not fully automatic. Moreover, patterns in their approach were still only n-grams. A similar research was reported by Uchino et al. [25]. They used basic phrase templates to automatically expand the number of template patterns and applied them to machine translation. They also focused only on n-gram based patterns. Research tackling patterns more sophisticated than n-grams was done by Talukdar et al. [24]. They proposed a context pattern induction method for entity extraction. However, in their research the seed word set was provided manually and the extraction limited to the patterns neighboring the seed words. Therefore the patterns in their research were limited to n-grams separated with one word inside the pattern. Moreover, their system disregarded grammatical information. Espresso, a system using grammatical information in pattern extraction was reported by Pantel and Pennacchotti [14]. Espresso used generic patterns to automatically obtain semantic relations between entities. However, although the patterns Espresso used were not limited to n-grams, they were very generic (e.g. is-a or part-of patterns) and were provided to the system manually. An idea close to ours, called “skipgrams” was proposed Guthrie et al. [23]. The idea assumed that “skips” could be put between elements of n-grams (similar to wildcards in SPEC). However, they focused only on bigrams and trigrams. Moreover, they assumed that n-gram elements could be separated at most by 4 skips, which makes the extraction of patterns shown in Introduction impossible. In comparison with the mentioned methods, our method is advantageous in several ways. Firstly, we aimed to fully automatize the process of generation of potential patterns and extraction of actual patterns. Secondly, we deal with patterns more sophisticated than n-grams, generic separated patterns or skipgrams.

3. SPEC - SYSTEM DESCRIPTION

This section contains detailed description of SPEC, or *Sentence Pattern Extraction arChitecturte*. In the sections below we describe the system sub-procedures. This includes corpus preprocessing, generation of all possible patterns, extraction of frequent patterns and post-processing. By a “corpus” we consider any collection of sentences or instances. It can be very large, containing thousands of sentences, or small consisting of only several or several dozen sentences. In any case SPEC automatically extracts frequent sentence patterns distinguishable for the corpus. In the assumption, the larger and the more coherent the original corpus is, the more frequent patterns will be extracted.

3.1 Corpus Preprocessing

SPEC was designed to deal with any not preprocessed raw corpora, as long as the lexical form of the language consists of smaller distinguishable parts, like letters, or characters. This makes SPEC capable to deal with corpora written in any type of language, including analytic languages (like English or Mandarin Chinese), agglutinative languages (like Japanese, Korean or Turkish), or even polysynthetic languages like Ainu, in their both spaced and non-spaced form. However, in the Pattern Generation sub-procedure, SPEC creates a very large number of temporary patterns (all possible ordered combinations of sentence elements). Therefore, considering the processing time, to avoid extensive combinatorial explosion, as a default we will assume that the corpus is at least spaced. Other relevant optional preprocessing might include part-of-speech (POS) tagging, dependency relation tagging or any other additional information as long as there exist sufficient tools. Three examples of preprocessing with and without POS tagging are presented in Table 1 for a sentence in Japanese[2]. The sentence in the example was spaced and tagged with MeCab [10], a standard POS tagger for Japanese.

Sentence:	今日はなんて気持ちいい日なんだ！
Transliteration:	*Kyōwanantekimochiiihinanda!*
Meaning:	Today TOP what pleasant day COP EXCL
Translation:	What a pleasant day it is today!
	Preprocessing examples
1. Words:	*Kyō wa nante kimochi ii hi nanda !*
2. POS:	N TOP ADV N ADJ N COP EXCL
3.Words+POS:	*Kyō*[N] *wa*[TOP] *nante*[ADV] *kimochi*[N] *ii*[ADJ] *hi*[N] *nanda*[COP] *!*[EXCL]

TABLE 1: Three examples of preprocessing of a sentence in Japanese with and without POS tagging; N = noun, TOP = topic marker, ADV = adverbial particle, ADJ = adjective, COP = copula, INT = interjection, EXCL = exclamative mark.

3.2 Pattern Generation

In this procedure SPEC generates all possible combinations of patterns from a sentence. Various algorithms have been proposed for creating combinations. Some use iteration loops, other use recursion. As processing speed is a crucial factor when dealing with reasonable size corpora, we designed two versions of a module to perform this procedure. The first version was designed to use one of the officially available iteration based algorithms for combination generation. In the second version we used a recursion algorithm designed especially for the task. Below we describe both versions. In section 4 we perform a speed test on four different iterative algorithms to choose the fastest one. The version of SPEC based on the fastest iterative algorithm is confronted later with the recursive version.

Iteration Based Algorithm

Generation of All Combinations from Sentence Elements

In this sub-procedure, the system generates ordered non-repeated combinations from the elements of the sentence. In every *n*-element sentence there is *k*-number of combination groups, such as $1 \leq k \leq n$, where *k* represents all *k*-element combinations being a subset of *n*. The number of combinations generated for one *k*-element group of combinations is equal to binomial coefficient, represented in equation 1. In this procedure we create all combinations for all values of *k* from the range of *{1,…,n}*. Therefore the number of all combinations is equal to the sum of all combinations from all *k*-element groups of combinations, like in the equation 2.

[2] Japanese is a non-spaced agglutinative language.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

$$\sum_{k=1}^{n} \binom{n}{k} = \frac{n!}{1!(n-1)!} + \frac{n!}{2!(n-2)!} + ... + \frac{n!}{n!(n-n)!} = 2^n - 1 \tag{2}$$

Ordering of Combinations

In mathematics, combinations are groups of unordered elements. Therefore, using only the above algorithm, we would obtain patterns with randomized order of sentence elements, which would make further sentence querying impossible. To avoid randomization of sentence elements and preserve the sentence order we needed to sort each time the elements of a combination after the combination has been generated. To do that we used automatically generated double hash maps. Firstly, all elements of the original input sentence are assigned ordered numbers (1, 2, 3...). After a combination is generated, elements of this combination are re-assigned numbers corresponding to the numbers assigned to the original sentence elements. The new set of numbers is sorted. Then the sorted list of numbers is re-mapped on original sentence elements (words) using the first hash. This provides the appropriate order of combination elements consistent with the order of elements in the original sentence. See Figure 1 for details of this part of the procedure.

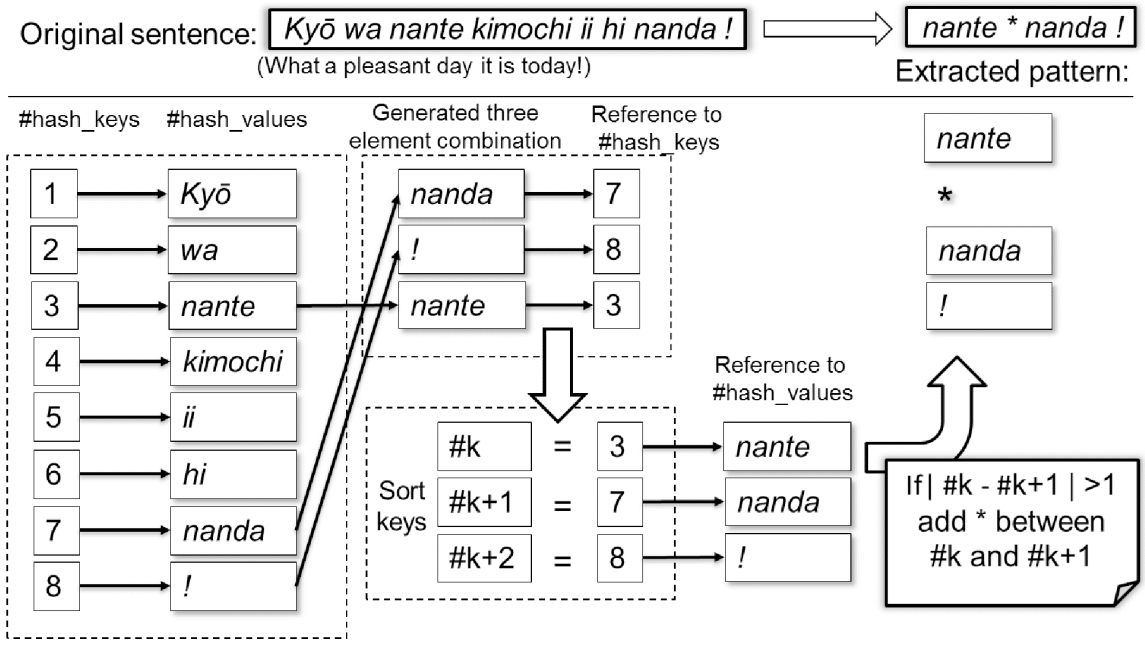

FIGURE 1: The procedure for sorting combination elements with automatically generated hash maps.

Insertion of Wildcard

In this stage the elements of a pattern are already sorted, however, to perform effective queries to a corpus we would also need to specify whether the elements appear next to each other or are separated by a distance. In practice, we need to place a wildcard between all non-subsequent elements. We solved this using one simple heuristic rule. If absolute difference of hash keys assigned to the two subsequent elements of a combination is higher than 1, we add a wildcard between them. This way we obtain a set of ordered combinations of sentence elements with wildcards placed between non subsequent elements. All parts of this procedure: generation of

combinations, sorting of elements using automatically generated hash maps and wildcard insertion, are represented in Figure 1.

Recursion Based Algorithm
The recursion based algorithm is a coroutine-type recursive generator, which produces a new item in a list each time it is called. It returns all the elements in the list passed into it, unless the previous element is the same (this removes adjacent wildcards). The algorithm flow goes as follows. Firstly, it writes out all possible combinations of the word list by continuously replacing subsequent words with wildcards. This operation is recursively called by itself, beginning from 0 wildcards to the overall number of sentence elements. The algorithm goes through all positions in the word list (sentence) starting from the beginning and places a wildcard there. This prevents infinite loops, since, if it goes beyond the end of the list, it will fall through without executing the loop. The original value is saved off and a wildcard is placed at this position. Then it calls itself recursively on the next index and with one less wildcard left to place. The list is restored to its original position and the operation is repeated till there is no more wildcards to place. Finally adjacent wildcards are removed and output is written to files. The whole procedure is performed for all sentences in the corpus.

3.3 Pattern Extraction and Pattern Statistics Calculation

In this sub-procedure SPEC uses all original patterns generated in the previous procedure to extract frequent patterns appearing in a given corpus and calculates their statistics. The statistics calculated include *number of pattern occurrences (O)*, *pattern occurrence frequency (PF)* and *pattern weight (W)*.

Number of pattern occurrences (O) represents the number of all occurrences of a certain *k*-long pattern in a given corpus.

Pattern occurrence frequency (PF) represents the number of all occurrences of a *k*-long pattern within a corpus divided by the number of all *k*-long patterns that appeared more than once *(Ak)*. See formula 3.

Pattern weight (W) is a multiplication of the length of *k* and *PF*. See formula 4.

$$PF_k = \frac{O}{A_k} \tag{3}$$

$$W_k = k * PF_k \tag{4}$$

The general pragmatic rule which applies here says that the longer the pattern is (length *k*), and the more often it appears in the corpus (occurrence *O*), the more specific and representative it is for the corpus (weight *W*). The pattern frequency calculation can be performed as a part of the previous procedure (pattern generation) when dealing with only one corpus. However, an often need in tasks like document classification is to obtain a set of patterns from a training (often smaller) corpus and perform pattern matching on a larger corpus. For example, in lexicon expansion one uses seed patterns to find out new vocabulary and phrases appearing within the patterns. We assumed SPEC should be able to deal with both, single and multiple corpus case. Therefore we needed to retain the ability to generate patterns from one (given) corpus and match them to another (target) corpus. Separating pattern generation and pattern extraction procedures also allows cross-reference of two or more corpora (a case when both are given and target corpora). This also allows performing the extraction on all corpora concurrently, e.g., using *fork* or *thread* functions for parallel programming, which shortens processing time. Finally, by making the system module-based (all sub-procedures are created as separate modules), each sub-procedure can be thoroughly evaluated, improved, expanded, or substituted with more efficient one when needed.

3.4 Post Processing

In the post-processing phase SPEC performs simple analysis of patterns extracted from the given corpus to provide its basic pragmatic specifications. We use the term "pragmatic specifications" in a similar way to Burkhanov, who generally includes here indications and examples of usage [2]. The post-processing is done differently for: 1) one given/target corpus and 2) a set of two or more corpora.

One Corpus Case

The post-processing of one corpus is done as follows. Firstly, all patterns that appeared only once are filtered out and deleted. This is done to eliminate quasi-patterns. A quasi-pattern is a pattern, which was created from a sentence in the combinatorial explosion-based process of pattern generation, but was not found elsewhere in the rest of the corpus. In practice, it means that it is not a frequently used sentence pattern and therefore keeping it would bias the results. The patterns that appeared more than once are grouped according to pattern length (number of elements a pattern consists of). The patterns are also indexed within groups using initial pattern letters as indices. In this form the patterns can be used in further analysis.

Two/Multiple Corpora Case

In many NLP tasks, especially those taking advantage of machine learning methods, it is often necessary to obtain lists of two distinctive sets of features [5]. This refers to all sorts of text classification domains, like spam filtering, sentiment and affect analysis [13] or even novel ones, like cyber-bullying detection [16]. In the post-processing procedure SPEC refines the patterns extracted for several corpora to filter out the ones that appear exclusively in each corpus and the ones which occurrences repeat in several corpora. The post-processing for two corpora is done as follows. Firstly, SPEC deletes only those quasi-patterns that appeared only once in both corpora. For all patterns which appeared more than once in at least one corpus SPEC calculates for which of the two corpora they are more representative, applying the notions of *O*, *PF* and *W* defined in section **3.3**. Overall diagram of the SPEC system is represented in Figure 2.

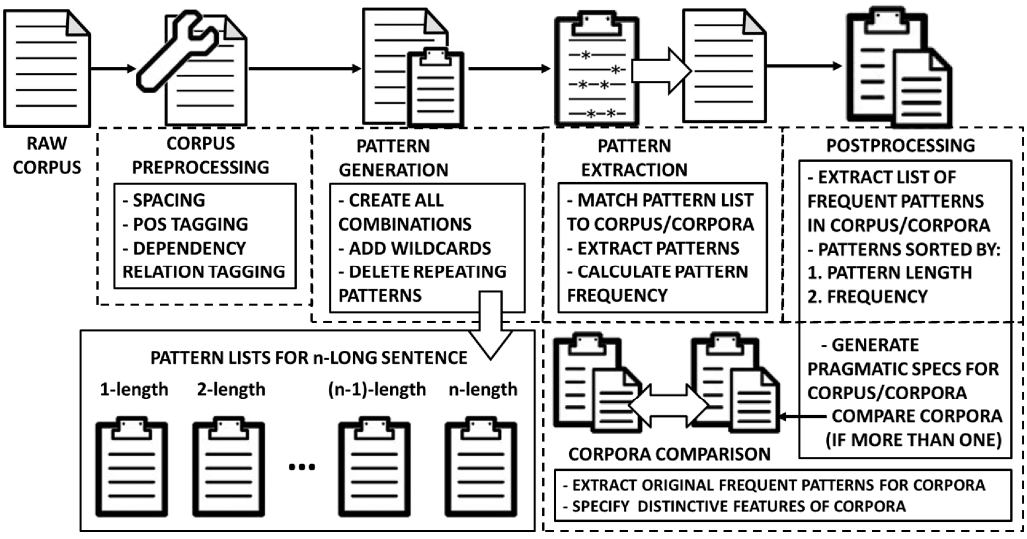

FIGURE 2: General diagram of the SPEC system.

4. EXPERIMENTS

This section describes experiments we performed to evaluate different aspects of SPEC. As time is a crucial issue in systems using combinatorial approach we begin with experiments on processing speed. Next, we perform a detailed analysis of patterns extracted by SPEC with comparison to a usual n-gram approach.

4.1 Test Sets

The experiments are performed on two different test sets. Firstly, as was shown by Potts et al. [15] and Ptaszynski et al. [17], pragmatic differences are the most visible in a comparison of emotional and neutral language. Therefore, we used a set of emotive and non-emotive utterances in Japanese collected by the latter. There are 38 emotive utterances and 39 non-emotive utterances in the set.

4.2 Experiments with Processing Time

All experiments were conducted on a PC with the following specifications. Processor: Intel Core i7 980X; Memory: 24 GB RAM; Storage: SSD 256 GB; OS: Linux Fedora 14 64bit.

Speed Test of Iterative Algorithms for Combination Generation

In the first experiment we compared speed in generating combinations for four officially available algorithms. As most of SPEC procedures are written in Perl, we used algorithms designed for this programming language.

Math::Combinatorics (M::C) is a Perl module designed by Allen Day. It provides a pure-perl implementation of functions like combination, permutation, factorial, etc. in both functional and object-oriented form. The module is available at: http://search.cpan.org/~allenday/Math-Combinatorics-0.09/

Algorithm::Combinatorics (A::C) is also a Perl module for generation of combinatorial sequences, designed by Xavier Noria. The algorithms used in this module were based on professional literature [8]. The module is available at: http://search.cpan.org/~fxn/Algorithm-Combinatorics/

Algorithm α is a subroutine based iteration algorithm keeping track of a list of indices.

Algorithm β is also a subroutine based iteration algorithm, although optimized. Both α and β algorithms are available on PerlMonks, a website for Perl programming community, at: http://www.perlmonks.org/?node id=371228

The above four algorithms were applied in a simple task of generating all combinations from a list containing all letters of alphabet[3]. Processing time was calculated separately for each *k*-long group of combinations. The test was taken ten times for each group. This provided 260 tests for each algorithm, giving over a thousand of overall number of tests. For each group of tests the highest and lowest outliers were excluded and the averages of processing times were calculated. The results for all four tested algorithms are represented in Figure 3.

Iteration Algorithm vs. Recursion Algorithm

In the second experiment we compared two versions of pattern generation procedure, based on the fastest iteration algorithm and recursion algorithm described in section **3.2**. The task was to generate from the same list (alphabet letters) not only combinations, but the whole patterns, with sorted elements and wildcards included. Here the time was calculated not for each combination group but for the whole process of generating all patterns. The test was also taken ten times and the highest and lowest outliers excluded. The averages of results are represented in table 2.

[3] 26 letters: ”a b c d e f g h i j k l m n o q p r s t u w v x y z”.

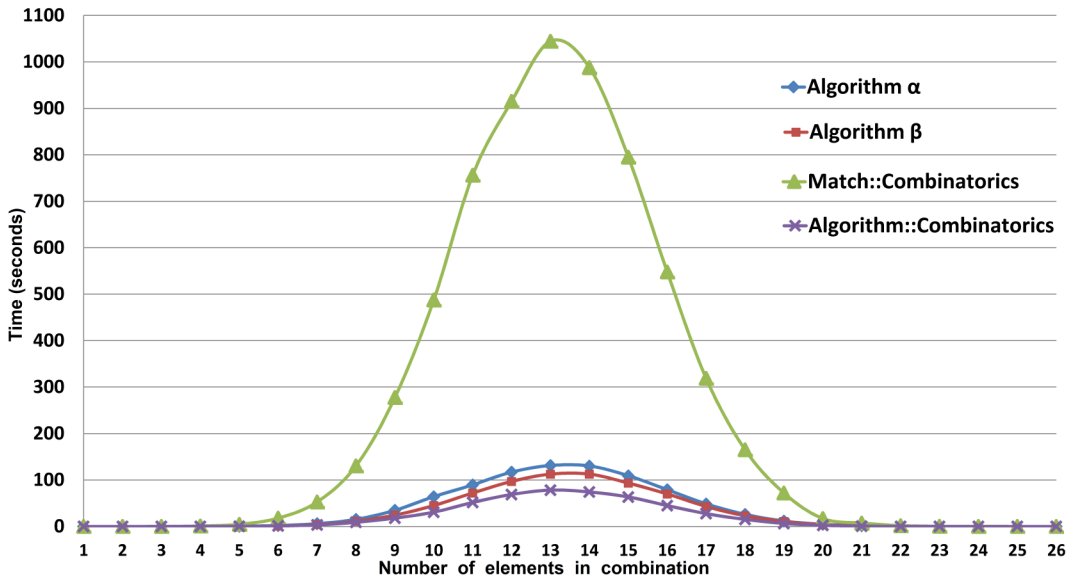

FIGURE 3: Graph showing time scores of all four tested algorithms.

Algorithm	Processing times (min., sec.)
Recursion based	24 min., 50.801 sec.
Iteration based	45 min., 56.125 sec.

TABLE 2: Averaged results of processing time compared between recursive and iterative version of pattern generation procedure.

4.3 Experiments with Pattern Analysis

N-gram Approach vs. Pattern Approach

By extracting frequent sentence patterns from corpora, SPEC builds a sentence pattern based language model from scratch for any corpus. An approach usually applied in language modeling is n-gram approach [1, 3, 19]. Therefore, for a comparison we used n-gram approach. In the experiment we calculated the number of extracted frequent patterns and compared it to the number of frequent n-grams. However, n-grams are also types of patterns and SPEC would generate n-grams as well. Therefore in this experiment we compared the number of extracted n-grams with the number of only non-n-gram patterns. As we assumed, this should show how effective is the pattern based method over n-grams. It might seem that the pattern based method would always perform better, since it generates incomparably larger numbers of patterns. However, in the evaluation we used only those patterns, which appeared in corpus more than once. This means we filter out all potential (or quasi) patterns leaving only the frequent ones. Thus the actual differences might not be of that high order. We also verified whether the differences between the numbers of frequent patterns and frequent n-grams were statistically significant using Student's T-test. Since the compared groups are of the same type (*k*-long patterns) we decided to use paired version of T-test. If the differences were small and not significant, it would mean that the traditional n-gram approach is equally good while much faster than the proposed pattern approach. We also compared the patterns in two different ways. Quantitative, referring to the overall number of patterns per group and Qualitative, in which we also compared how frequent were the patterns within a group. This way, a group with smaller number of very frequent patterns, could score higher than a group of many modestly frequent patterns.

5. RESULTS AND DISCUSSION

In the processing speed experiment, in which we compared four combinatorial algorithms, the fastest score was achieved by Algorithm::Combinatorics. The worst was Math::Combinatorics. Other two algorithms achieved similar results. However, none of them was faster than A::C. Therefore we used this algorithm in iterative version of SPEC.

Next, we compared the processing time of pattern generating procedure for iteration and recursion based algorithms. The latter one was over two times faster (see Table 2). Although the iterative algorithm itself is fast, additional operations performed after generating the combinations, such as sorting of elements and insertion of wildcards, influence the overall processing time. Therefore for the task of generating all combinations, it is more efficient to use the recursion based algorithm. The further analysis of patterns, however, revealed that it is not always necessary to create all patterns. The experiment showed that generating up to 5-element combinations is sufficient to find all frequent language patterns from a corpus (see Figure 4). This discovery, when confirmed on larger datasets, on corpora of different languages and domains, should provide some deep insights about the nature of structured language patterns, like compound nouns, collocations or even conversational strategies, which would contribute greatly to the field of Corpus Pragmatics.

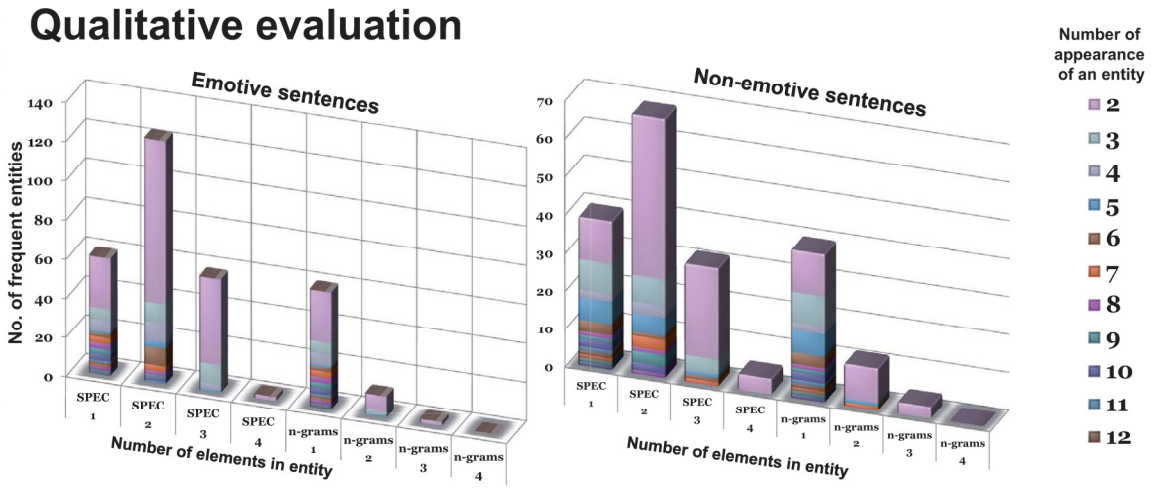

FIGURE 4: Graphs showing differences in numbers of frequent entities (patterns or n-grams) extracted from two datasets.

Except the discovery about the length of combinations sufficient for extraction of frequent patterns, quantitative analysis of patterns extracted from datasets revealed the following. While the number of frequent n-grams was decreasing rapidly with the increase in number of elements, the number of patterns increased for 2-element patterns and then gradually decreased, providing approximately 5 to 20 times more frequent patterns for emotive utterances and 5 to 10 times more patterns for non-emotive utterances (see Table 3). Whether the dataset domain (emotiveness) and language (Japanese) were influential on the overall number of patterns should be an object of further study, however, were able to prove that the pattern based approach does provide more frequent patterns in both of the examined cases. Moreover, qualitative analysis revealed, that n-grams appeared usually with the low occurrence and, excluding unigrams, there were almost no n-grams of higher occurrence then two. On the other hand, for patterns, about one third of the extracted patterns was of occurrence higher then 2 (3 or more). This proves that pattern based language analysis should allow more detailed analysis of language structures than the traditional n-gram approach. Finally, all results were statistically significant on a standard 5% level.

Data sets	n-grams	patterns (without n-grams)	all patterns
	2-element entities		
Emotive sentences	11	113	124
Non-emotive sentences	11	57	68
	3-element entities		
Emotive sentences	2	56	58
Non-emotive sentences	3	28	31
	4-element entities		
Emotive sentences	0	4	4
Non-emotive sentences	0	4	4

TABLE 3: Number of frequent entities extracted (n-grams, non-n-gram patterns and all patterns) from two datasets (collections of emotive and non-emotive sentences). Results presented separately for 2, 3, and 4 element entities.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented SPEC, or *Sentence Pattern Extraction arChitecturte*. The system extracts all frequent sentence patterns from corpora. The extracted patterns are more sophisticated than the ones obtained in a usual n-gram approach, as SPEC does not assume that two subsequent elements of a pattern appear subsequently also in the sentence. SPEC firstly generates all combinations of possible patterns and calculates their pattern statistics (number of occurrences, frequency and weights). SPEC is capable of processing corpora written in any language, as long as they are minimally preprocessed (spacing, POS tagging, dependency structure, etc.). Experiments showed that the method extracts significantly more frequent patterns than the usual n-gram approach. There are two issues needing attention in the near future. Firstly, to make this approach scalable to large corpora (several thousands of sentences of different length), the algorithms need to be further optimized to generate and extract patterns in a faster manner. In pattern generation, a further study in the longest sufficient pattern length will be necessary. Also, applying high performance computing techniques, such as parallel computing should provide much decrease in processing time. Another issue is noise. Although there have been no coherent definition so far of what a noise is in n-gram models, it would be naive to assume that all patterns are always valuable for all applications of the method. One method to deal with this issue should be raising the threshold of frequent patterns (to 3 or higher). Finally, in the near future we plan to apply SPEC to other NLP tasks, such as spam classification, user detection, sentiment analysis [13], cyberbullying detection [16], or lexicon expansion for affect analysis [17] to evaluate SPEC in practice.

Acknowledgments

This research was supported by (JSPS) KAKENHI Grant-in-Aid for JSPS Fellows (Project Number: 22-00358). Authors thank Tyson Roberts and Jacek Maciejewski for their invaluable help with optimizing SPEC algorithms. SPEC repositories are available freely at: http://arakilab.media.eng.hokudai.ac.jp/~ptaszynski/research.htm

REFERENCES

[1] P. F. Brown, P. V. de Souza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. “Class-based n-gram models of natural language”. Computational Linguistics, Vol. 18, No. 4 (December 1992), 467-479, 1992.

[2] Burkhanov. “Pragmatic specifications: Usage indications, labels, examples; dictionaries of style, dictionaries of collocations”, In Piet van Sterkenburg (Ed.). A practical guide to lexicography, John Benjamins Publishing Company, 2003.

[3] S. Chen, J. Goodman, “An empirical study of smoothing techniques for language modeling”, Comp. Speech & Language, Vol. 13, Issue 4, pp. 359-393, 1999.

[4] N. Constant, C. Davis, C. Potts and F. Schwarz, “The pragmatics of expressive content: Evidence from large corpora”. Sprache und Datenverarbeitung, 33(1-2):5-21, 2009.

[5] G. Forman. “An extensive empirical study of feature selection metrics for text classification”. J. Mach. Learn. Res., 3 pp. 1289-1305, 2003.

[6] P. H. Grice, Studies in the Way of Words. Cambridge (MA): Harvard University Press, 1989.

[7] D. Knight, and S. Adolphs, “Multi-modal corpus pragmatics: The case of active listenership”, Pragmatics and Corpus Linguistics, pp. 175-190, Berlin, New York (Mouton de Gruyter), 2008.

[8] D. E. Knuth, The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. Addison Wesley Professional, 2005.

[9] K. Krippendorff, “Combinatorial Explosion”, Web Dictionary of Cybernetics and Systems. Princia Cybernetica Web.

[10] T. Kudo. MeCab: Yet Another Part-of-Speech and Morphological Analyzer, 2001. http://mecab.sourceforge.net/ [July 27, 2011].

[11] S. C. Levinson, Pragmatics. Cambridge University Press, 1983.

[12] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. “Text classification using string kernels”, The Journal of Machine Learning Research, 2, pp. 419-444, 2002.

[13] B. Pang, L. Lee, S. Vaithyanathan. “Thumbs up?: sentiment classification using machine learning techniques”. In Proc. of EMNLP'02, pp. 79-86, 2002.

[14] P. Pantel and M. Pennacchiotti, “Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations”, In Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pp. 113-120, 2006.

[15] C. Potts and F. Schwarz. “Exclamatives and heightened emotion: Extracting pragmatic generalizations from large corpora”. Ms., UMass Amherst, 2008.

[16] M. Ptaszynski, P. Dybala, T. Matsuba, F. Masui, R. Rzepka, K. Araki and Y. Momouchi, “In the Service of Online Order: Tackling Cyber-Bullying with Machine Learning and Affect Analysis”, International Journal of Computational Linguistics Research, Vol. 1 , Issue 3, pp. 135-154, 2010.

[17] M. Ptaszynski, P. Dybala, R. Rzepka K. and Araki, “Affecting Corpora: Experiments with Automatic Affect Annotation System - A Case Study of the 2channel Forum”, Proceedings of PACLING-09, pp. 223-228, 2009.

[18] E. Riloff, “Automatically Generating Extraction Patterns from Untagged Text”, In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), pp. 1044-1049, 1996.

[19] B. Roark, M. Saraclar, M. Collins, “Discriminative n-gram language modeling”, Computer Speech & Language, Vol. 21, Issue 2, pp. 373-392, 2007.

[20] K. Sasai, “The Structure of Modern Japanese Exclamatory Sentences: On the Structure of the *Nanto*-Type Sentence”. Studies in the Japanese Language, Vol, 2, No. 1, pp. 16-31, 2006.

[21] F. Sebastiani. “Machine learning in automated text categorization”. ACM Comput. Surv., 34(1), pp. 1-47, 2002.

[22] C. E. Shannon, “A Mathematical Theory of Communication”, The Bell System Technical Journal, Vol. 27, pp. 379-423 (623-656), 1948.

[23] D. Guthrie, B. Allison, W. Liu, L. Guthrie, Y. Wilks, Y. “A Closer Look at Skip-gram Modelling”. In Proc. Fifth International Conference on Language, Resources and Evaluation (LREC'06), pp. 1222-1225, 2006.

[24] P. P. Talukdar, T. Brants, M. Liberman and F. Pereira, “A Context Pattern Induction Method for Named Entity Extraction”, In Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X), pp. 141-148, 2006.

[26] H. Uchino, S. Shirai, S. Ikehara, M. Shintami, “Automatic Extraction of Template Patterns Using n-gram with Tokens” [in Japanese], IEICE Technical Report on Natural Language Understanding and Models of Communication, 96(157), pp. 63-68, 1996.