Learning and Exploiting Non-Consecutive String Patterns for Information Extraction

Yunbo Cao
Microsoft Research Asia
5F Sigma Center, No.49
Zhichun Road, Haidian
Beijing, China, 100080
i-yucao@microsoft.com

Hang Li
Microsoft Research Asia
5F Sigma Center, No.49
Zhichun Road, Haidian
Beijing, China, 100080
hangli@microsoft.com

Shenjie Li
Computer Science Department
Tsinghua University
Haidian
Beijing, China, 100084

Abstract

This paper is concerned with the problem of learning and exploiting string patterns in natural language processing, particularly information extraction. We propose a new algorithm for learning such patterns. Our algorithm is novel in that it can learn *non-consecutive patterns with constraints*, which are necessary for information extraction. Specifically, it employs an extended version of the so-called *apriori algorithm* at the pattern generation step. Our experimental results indicate that in information extraction the use of non-consecutive patterns with constraints is *significantly* better than the use of only consecutive patterns.

1 Introduction

We consider here the problem of learning and using string patterns, particularly *non-consecutive* patterns， in information extraction.

In learning, for example, given the instances "<company> today announced the worldwide availability of <product>", "<company> today announced the immediate availability of <product>", etc, we acquire non-consecutive patterns like "<company> today announced the \w+ availability of <product>", where '\w+' is a wildcard denoting a skip of at least one word. We refer to the patterns as 'non-consecutive patterns'. Note that 'consecutive patterns' are special cases of 'non-consecutive patterns'. In extraction, we use the acquired patterns to match the strings in new instances and extract from the matched strings information on which company has released what product.

Methods for learning and using only *consecutive* patterns for information extraction have been proposed (e.g., Brin 1998; Ravichandran and Hovy 2002). The coverage of such patterns is small, however, as they do not contain generalization.

In this paper, we propose a new algorithm which can *accurately* and *efficiently* learn non-consecutive patterns with constraints. Our algorithm consists of two steps: pattern generation and pattern ranking. At the first step, it creates all the possible patterns which cover the positive instances. At the second step, it ranks the patterns according to their precision scores using both the positive and negative instances.

Our algorithm is especially novel in that it employs an extended version of the *apriori* algorithm to accurately and efficiently acquire patterns. The apriori algorithm was proposed for learning associations in the field of data or text mining. We think that it is the first time that it is used for the information extraction task. Furthermore, the apriori algorithm is extended here for learning patterns with *constraints*. We propose three constraints necessary for accurately acquiring non-consecutive patterns. We prove that even on the constraints, the so-called *apriori* (or *anti-monotonicity*) property still holds.

We applied the proposed algorithm to product-release information extraction from the web sites of IT companies. We also applied it to question answering regarding inventions. Experimental results indicate that the use of non-consecutive patterns with constraints *significantly* outperforms the use of only consecutive patterns in information extraction. Experimental results also indicate that the *constraints* we have defined are necessary for accurate extraction.

2 Related Work

2.1 Information Extraction Using String Patterns

A straightforward approach to learn natural language patterns would be to syntactically parse sentences and acquire sentential or phrasal patterns from the obtained parsed trees (e.g., Lin and Pantel, 2001; Sudo et al, 2001). Another approach would be to discover string patterns by using syntactic and semantic constraints (e.g., Huffman 1995; Soderland et al, 1995; Riloff, 1996). The two approaches are in general costly in development.

Methods for directly acquiring *consecutive* patterns from surface word strings have been proposed.

For example, Brin (1998) proposed learning and using *consecutive* patterns for extracting information on certain pairs such as (<author>, <book>) pairs. They conducted extraction from web data through a bootstrapping process.

For another example, Ravichandran and Hovy (2002) proposed learning and using *consecutive* patterns for extracting information on <question, answer> pairs in question answering. For example, they extracted patterns like '<person> was born in <year>' for answering a question regarding the birth year of a person. Their method consisted of two steps: pattern generation and pattern ranking. They used a *suffix tree* to store all the possible string patterns at pattern generation.

2.2 Sequential Data Mining

Agrawal and Srikant (1994) proposed employing the apriori algorithm for mining patterns from sequential data. Each data sequence is a list of transactions ordered by transaction-time. Subsequently, Srikant and Agrawal (1996) proposed extending the algorithm by incorporating time and taxonomy constraints. Lent et al (1997) proposed using the apriori algorithm for mining phrasal patterns in order to discover trends in a text database.

For other work on sequential data mining, see (e.g., Mannila and Toivonen, 1996; Ahonen et al, 1998).

The apriori algorithm was mainly used for learning *associations* between data items in data mining, or words in text mining. It was not used for learning patterns necessary for information extraction. Note that there are some clear differences between information extraction and text mining. For example, information extraction is generally concerned with more *complex* patterns than text mining. Information extraction generally needs annotated data for training, while text mining does not necessarily need.

3 Problem

In learning, given 'positive and negative training instances' as those in Figure 1, we are to acquire patterns as those in Figure 2. In extraction, we use the acquired patterns to extract information from 'test instances' as that in Figure 1. Note that there are negative test instances from which no information can be extracted.

Positive training instances: <company> Microsoft Corp. </company> today announced the immediate availability of <product> Microsoft Internet Explorer Plus </product>, the eagerly awaited retail version of Internet Explorer 4.0. <company> Microsoft Corp. </company> today announced the availability of <product> Microsoft Visual J++ 6.0 Technology Preview 2</product>, a beta release of the next version of the industry's most widely used development system for Java. <company> Microsoft Corp. </company> today announced the immediate, free availability of <product> Microsoft Visual InterDev 6.0 March pre-release </product>, a preview of the new version of the leading team-based Web development system for rapidly building data-driven Web applications.
Negative training instance: <company> Microsoft Corp. </company> today announced the availability of an expanded selection of Web-based training through its independent training providers.
Positive test instance: <company> Microsoft Corp. </company> today announced the immediate worldwide availability of Microsoft Office 60 Minute Intranet Kit version 2.0, downloadable for free (connect-time charges may apply) from the Office intranet Web site located at http://www.microsoft.com/office/intranet/.

Figure 1: Training and Test Data

The example in Figure 1 is concerned with product release information extraction. Positive instances contain information on "<company> has released <product>". (We replace here specific names such as '<company>Microsoft</company>' with variables such as '<company>'). Negative instances contain information on <company>, but it is not about product release. The goal is to extract from the positive test instances information on "<company> has released <product>", more precisely, the name of <product>, e.g., Micro-

soft's "Microsoft Office 60 Minute Intranet Kit version 2.0". That is to say, we assume here that in extraction <company> has already been identified, but <product> has not. Hereafter, we will sometimes refer to <product> as 'anchor'.

Consecutive patterns:

<company> today announced the immediate availability of <product>,

<company> today announced the availability of <product>,

<company> today announced the immediate, free availability of <product>,

Non-consecutive patterns:

<company> today announced the {\w +3} availability of <product>,

Figure 2: Patterns

Using the existing methods (e.g., Brin 1998; Ravichandran and Hovy 2002), one can obtain consecutive patterns as those in Figure 2. The coverage of such patterns is small, however, as they do not contain *generalization*. For example, using the patterns, one cannot extract the information in the test instance in Figure 1. It is obvious that the words 'immediate' and 'free' are not necessary for the extraction, and thus it is desirable to learn and use non-consecutive patterns that can *skip* such kind of words. Figure 2 also shows an example of non-consecutive patterns. With the pattern, one can correctly extract the information in the instance. The question then is how to acquire non-consecutive patterns.

4 Our Method

Our method of learning *non-consecutive* (and also consecutive) patterns comprises of two steps: *pattern generation* and *pattern ranking*. At the first step, it creates all the possible patterns which cover the positive instances. At the second step, it ranks the patterns according to their precision scores using both the positive and negative instances. In extraction, it utilizes the ranked patterns to match strings and extracts the anchor information from the matched strings.

4.1 Pattern Generation

The input of pattern generation is a number of strings, and the output is a number of non-consecutive patterns that cover the input strings. We replace specific names with general variables (e.g., <company>, <product>). The non-consecutive patterns are unique in that they can have 'wildcards'. We use '\w +*n*' to denote a wildcard that skips at least one and at most *n* words. The non-consecutive patterns thus contain *generalization* of strings.

Algorithm

Learn-non-consecutive-pattern-with-constraints

1. S = set of input strings,
2. P_1 = set of words in S ;
3. **for** ($i = 2; i \le k; i++$){
4. P_i = **find-nonconsecutive-pattern**($P_{(i-1)}$, P_1);
5. **for each** ($p \in P_i$){
6. **if** (not **satisfy-constraints**(p))
7. remove p from P_i ;
8. **if** (p 's frequency is not larger than a threshold)
9. remove p from P_i ;
10. **if** (p does not contain <anchor>)
11. remove p from P_i ;
12. }
13. **if** (P_i is empty)
14. Goto line 16;
15. }
16. output $P = \bigcup_{j=2}^{i} P_j$;

Figure 3: Algorithm of Pattern Generation

find-non-consecutive-pattern($P_{(i-1)}$, P_1 **)**

1. **for each (** $p_{(i-1)} \in P_{(i-1)}$ **){**
2. **for each (** $p_1 \in P_1$ **){**
3. $p_i = p_{(i-1)} p_1$ **;**
4. **if** (p_i exists in S)
5. **put** p_i **into** P_i **;**
6. $p'_i = p_{(i-1)} \{\backslash w + n\} p_1$ **;**
7. **if** (p'_i exists in S)
8. **put** p'_i **into** P_i **;**
9. **}**
10. **}**
11. output P_i **;**

Figure 4: Sub-Algorithm of Pattern Generation

Figure 3 shows the algorithm which generates patterns satisfying the constraints described below. The algorithm is an extension of the apriori algorithm.

Let P_i denote the set of generated patterns in the *i*-th iteration ($1 \le i \le k$). Initially, let P_1 to be the set of words. Our algorithm recursively creates patterns in P_i by combining the patterns in $P_{(i-1)}$ and the words in P_1 . The algorithm comprises of two sub algorithms:

'find-non-consecutive-patterns' (Figure 4) and 'satisfy constraints'.

At lines 6 and 7 of Figure 3, we check if pattern p satisfies the constraints, *if not* we remove it from the set of patterns. At lines 8 and 9, we check if the frequency of pattern p is not larger than a threshold, if so we remove it from the set of patterns (the same as in apriori).

At line 3 of Figure 4, we concatenate patterns $p_{(i-1)}$ and p_1 into pattern p_i. At line 6 of Figure 4, we concatenate patterns $p_{(i-1)}$ and p_1 into pattern p'_i in which there is a wildcard of at least one word and at most n words between $p_{(i-1)}$ and p_1, where n is calculated with the input data.

In the algorithm *we treat a wildcard as a special word.* As a result, for example, the string '\w the book \w' is not the superstring of the string '\w the \w'.

Note that 'find *consecutive* patterns' becomes a special case of 'find *non-consecutive* patterns', if we remove lines 6, 7 and 8 in Figure 4. Also note that 'find non-consecutive patterns *without* constraints' becomes a special case of 'find non-consecutive patterns *with* constraints', if we remove lines 6 and 7 in Figure 3.

Three Constraints

We propose the use of three constraints necessary for accurately acquiring non-consecutive patterns for information extraction.

The first constraint is that there cannot be a wildcard immediately before or after an anchor. We call the constraint 'boundary constraint'. The constraint is *obligatory* for information extraction, since it is necessary to accurately determine the boundaries of an anchor (e.g., <product>). Without this constraint, pattern 1 in Figure 5 will be generated, and with the pattern the information in test instance 1 will be incorrectly extracted.

The second constraint is that the number of n in the wildcard '\w +n' in a context should not be larger than the largest number of words to be skipped in the same context in the training data. We call the constraint 'distance constraint'. Without this constraint, pattern 2 in Figure 5 will be generated, and the information in test instance 2 will be incorrectly extracted.

The third constraint is that 'an isolated function word' is prohibited. For example, in the pattern '\w+ the \w+', 'the' is an isolated function word. The rational behind the constraint is that a pattern should include content words and skip *isolated* function words. We call the constraint 'island constraint'. Without this constraint, pattern 3 in Figure 5 will be generated, and the information in test instance 3 will be incorrectly extracted.

Non-consecutive patterns without certain constraint:

1. <company> today announced the immediate availability {\w +3} <product>
2. <company> {\w +} today announced {\w +} deliver <product>.
3. <company> {\w +8} the {\w +13} of the <product> , the first

Test instances:

1. Microsoft Corp. today announced the immediate availability of Internet Explorer for no-charge download from the Internet.
2. Microsoft Corp. and Policy Management Systems Corp. (PMSC) today announced a plan in which the two companies will work together to deliver enterprise and electronic-commerce solutions based on the Microsoft Windows NT Server operating system and the BackOffice family of products.
3. Microsoft Corp. today provided attendees of the Consumer Electronics Show in Las Vegas with a demonstration of the Microsoft Entertainment Pack for the Windows CE operating system, the first game product to be released for the Windows CE-based handheld PC platform.

Incorrectly extracted <product> information:

1. *any substring* of 'of Internet Explorer for no-charge download from the Internet'
2. enterprise and electronic-commerce solutions based on the Microsoft Windows NT Server operating system and the BackOffice family of products
3. Microsoft Entertainment Pack for the Windows CE operating system

Figure 5: Patterns without Constraints

The use of the constraints also has a desirable effect of improving efficiency in learning, as it helps reduce the search space.

Theorem 1 below guarantees that our algorithm is able to find all the patterns which cover the input strings satisfying the constraints.

Definition 1 $\rho_b(s)$, $\rho_d(s)$ and $\rho_i(s)$ are properties of string s, such that

$$\rho_b(s) = \begin{cases} 1 \text{ if } s \text{ satisfies the boundary constraint} \\ 0 \text{ otherwise} \end{cases}$$

$$\rho_d(s) = \begin{cases} 1 \text{ if s satisfies the distance constraint} \\ 0 \text{ otherwise} \end{cases}$$

$$\rho_i(s) = \begin{cases} 1 \text{ if s satisfies the island constraint} \\ 0 \text{ otherwise} \end{cases}$$

Definition 2 (anti-monotonicity) Let *s* denote any string and let *t* denote any superstring of *s*. A property ρ of strings is anti-monotone, if $\rho(s)=0$ implies $\rho(t)=0$.[1]

Theorem 1 $\rho_b(s)$, $\rho_d(s)$ and $\rho_i(s)$ are anti-monotonic.

The proof of the theorem is omitted here due to the limitation of space.

4.2 Pattern Ranking

The patterns obtained at pattern generation are ranked based on their precisions, using both positive and negative instances, provided that the precisions are larger than a predetermined threshold.

Let *a* denote the number of instances matched to a pattern *p*, and let *c* denote the number of the instances matched to *p*, and at the same time the information in the instances can be correctly extracted. The precision of *p* is defined as $\frac{c}{a}$.

When a pattern *p* can match all the positive instances that a pattern *q* can match, we say *p* covers *q*. If *p* covers *q* and *p* is ranked before *q* in the ranked pattern list, we remove *q* from the list.

4.3 Extraction

Given a new instance, the ranked patterns are examined sequentially. With the pattern which matches the instance first, the anchor information is extracted. For example, with the non-consecutive pattern in Figure 2, one can extract from the test instance in Figure 1, the anchor (i.e., <product>) information: "Microsoft Office 60 Minute Intranet Kit version 2.0".

The matching of a string pattern to a string instance is performed in the left-to-right order. For example for the pattern '*x* \w+ *y* \w+':

> *First, x matches to its first occurrence in the string and y matches to all its occurrences in the sub-string after the first occurrence of x. If the matching fails, then x matches to its second occurrence and y matches to all its occurrences in the remaining sub-string. The matching continues.*

[1] In mathematics, anti-monotonicity is a more general notion.

5 Experimental Results

We conducted two experiments in order to test the effectiveness of our method. Specifically, we performed information extraction regarding product releases and inventions.

Experimental results indicate that for information extraction (at least for the problems investigated), the use of non-consecutive string patterns with constraints outperforms the use of consecutive string patterns alone and the constraints we propose are indeed needed.

Hereafter, we denote the method using non-consecutive string patterns and that using consecutive string patterns as NCP and CP, respectively.

5.1 Product Release Information Extraction

Many companies routinely publish information on product releases at their web sites. Automatically extracting such information has not only research values but also practical interests. This is exactly the problem we have investigated, which is also described above as an example.

We collected press release articles from the websites of five IT companies. From each article, we extracted the first sentence using heuristic rules (the data in Figure 1 are examples of them). Next, we asked two human annotators to assign labels on the extracted sentences. The sentences containing information on "<company> has released <product>" were annotated as positive instances. Specifically, the company names and the product names were assigned labels. The sentences containing information only on <company> were annotated as negative instances. Specifically, only the company names were assigned labels. Details of the data can be found in Table 1.

Table 1: Data for Product Release Information Extraction

Company	Num. of pos. data	Num. of neg. data
Company A	174	229
Company D	304	390
Company I	250	556
Company M	1004	2365
Company N	208	292

With the data of each company, we compared NCP against CP. To investigate the necessity of the use of the constraints, we also tested two additional methods. In the first method, we removed the distance constraint from NCP, in the second method, we removed the island constraint from NCP. We denote them as NDC and NIC, respectively. Note that the boundary constraint is an obligatory constraint.

We performed the experiments with *10-fold cross-validation.* The results obtained were thus those averaged over ten experimental trials. We evaluated the results in terms of precision, recall and f-measure. Let |D| denote the number of instances extracted by using the patterns. Let |F| denote the number of instances *correctly* extracted by using the patterns. Let |E| denote the number of instances that should be extracted. We define

$$\text{precision} = |F|/|D|$$

$$\text{recall} = |F|/|E|$$

$$\text{f-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

With different thresholds on the precisions at pattern ranking, we obtained results of different precisions and recalls on the test set. Figures 6-10 show the 'precision recall curves" for the five companies.

From the figures, we see that NCP *significantly* outperforms CP with respect to Companies D, I, M, and N. It performs as well as CP with respect to Company A. Furthermore, NCP performs better than both NDC and NIC in most cases (note that NDC and NIC should also be considered as our methods). The results indicate that the use of non-consecutive patterns is better. They also indicate that both the distance and island constraints are needed for reliable information extraction.

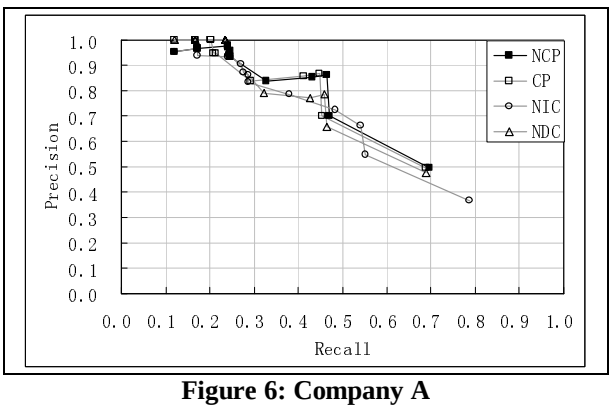

Figure 6: Company A

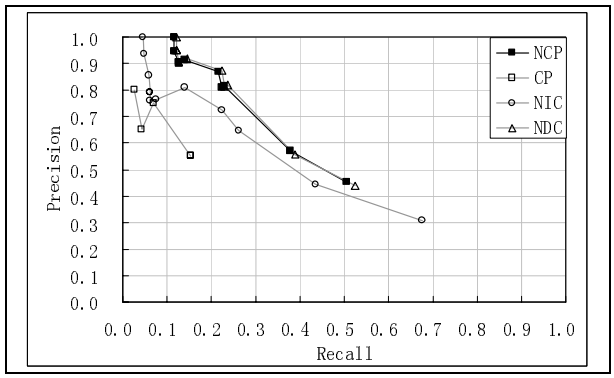

Figure 7: Company D

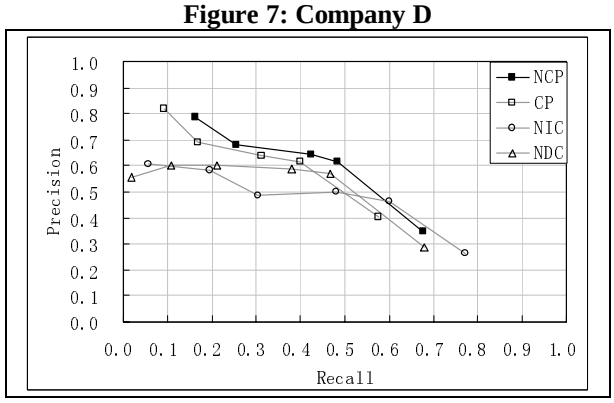

Figure 8: Company I

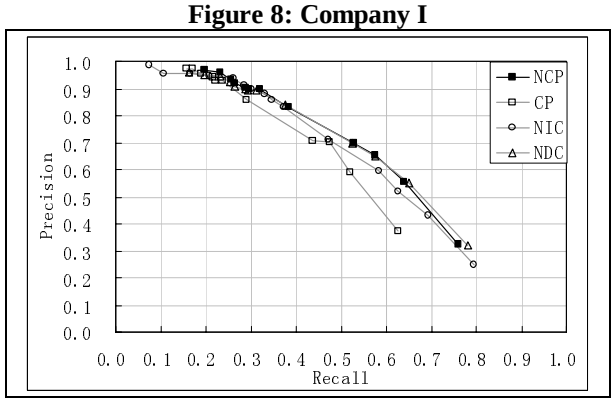

Figure 9: Company M

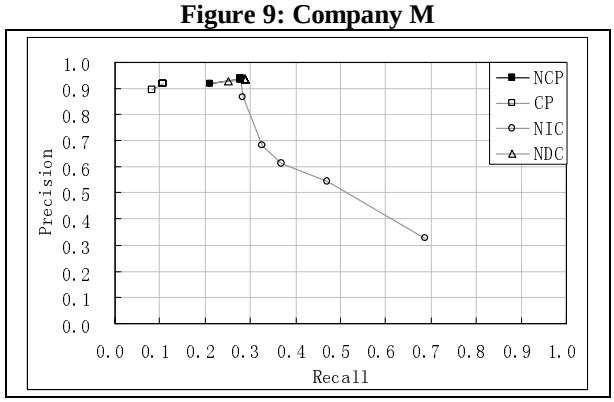

Figure 10: Company N

Table 2 shows the *highest* f-measure for each method. From the results, we see that NCP has the best performances in terms of highest f-measure in most of the cases except that of 'Company N'.

Table 2: Highest F-Measure (%)

	NCP	NDC	NIC	CP
Company A	**60.45**	57.97	59.49	59.09
Company D	**47.83**	47.68	43.78	24.16
Company I	**54.14**	51.32	52.26	48.43
Company M	**61.12**	60.98	58.91	56.67
Company N	42.96	44.12	**50.52**	18.97

NIC has the best f-measure for Company N. This is because without the island constraint more patterns could be acquired from the company's data, and the recall values turned out to be very high, as shown in Figure 10.

NCP:
1. <company> {\w +6} the availability of <product>,
2. <company> {\w +8} announced the {\w +5} availability of <product>.
3. <company> {\w +3} unveiled <product>, a
4. <company> today {\w +10} release of <product>, the
5. <company> {\w +5} announced the <product>, a

CP:
1. <company> today announced the availability of <product>,
2. <company> today announced the <product>, the
3. <company> today announced the immediate worldwide availability of <product>.
4. <company> today announced the release of the <product>, the
5. <company> today announced the <product>, a
6. <company> today unveiled <product>, a

Figure 11: Patterns with respect to Company M

Figure 11 shows examples of the patterns for Company M extracted by NCP and CP respectively. Since non-consecutive patterns include consecutive patterns, we omit the duplicated consecutive patterns from NCP in the figure. We see that NCP has more *generalized* patterns than CP, which contribute to the better performance of NCP.

Admittedly, the precisions of NCP are still not high enough. We investigated the reason and found that this was due to the limited coverage of the training data. We believe, therefore, that with more data being available the accuracies will be able to be further improved in the future.

5.2 Question Answering on Inventions

We conducted information extraction for question answering. More specifically, we conducted extraction on <question, answer> pairs. Given a <question> we can use the extracted <answer> to reply to the <question>. This experiment is similar to that in (Ravichandran and Hovy, 2002).

We selected four types of <question, answer> pairs related to inventions. They were <discovery, discoverer>, <inventor, invention>, <invention, inventor> and <invention, invention year> pairs.

We extracted data from the web. Table 3 shows the details of the data. First, we created <question, answer> pairs. For example, <McCormick, the mechanical reaper> is an <inventor, invention> pair. For each <question, answer> pair, we used a search engine[2] to search web pages containing both the <question> and the <answer>. From the top 50 returned pages, we extracted the sentences containing the <question> and the <answer>. We used them as positive instances for *pattern generation*. For each <question>, we also used the search engine to search web pages containing the <question>. From the top 50 returned pages, we extracted the sentences containing the <question>. We used them as instances (having both positive and negative instances) for *pattern ranking*.

Table 3: Data for Question Answering

Q/A type	Num. of Q/A pairs	Num. of data for generation	Num. of data for ranking
Discoverer	31	1280	97312
Invention	112	2412	184209
Inventor	112	2412	93960
Invention year	71	1154	57185

The name of a person can appear in data in various ways. For example, 'McCormick' can appear as 'Cyrus Hall McCormick', 'Cyrus McCormick', and 'Cyrus'. In the experiment, we normalized the names of persons (e.g., normalized the above names to 'McCormick'). We also assumed that any 3 or 4 digitals can be the expression of a year.

We performed pattern learning and extraction experiments using the data through *10-fold cross*-validation. The results obtained thus were averaged over ten trials.

We evaluated the results using the 'mean reciprocal rank (MRR)' measure (Voorhees, 2001), which was also used in (Ravichandran and Hovy, 2002). If the correct answer appears in the top 5 extracted answers, 'reciprocal rank' is defined as the reciprocal of the rank of the first correct answer; otherwise, 'reciprocal rank' is defined as zero. MRR is the mean of the re-

[2] We used Google (http://www.google.com).

as zero. MRR is the mean of the reciprocal ranks of all the questions.

We note that precision-recall is suitable for the evaluation in the first experiment and MRR is suitable for the evaluation in the second experiment. In the first experiment, extraction was performed in order to find all the information from a closed data set; while in the second experiment, extraction was performed so that whenever an answer was found, the search of an answer would be no longer needed.

Table 4: MRR Scores

Q/A type	NCP	CP
Discoverer	**0.776**	0.709
Invention	**0.375**	0.305
Inventor	**0.407**	0.360
Invention year	0.390	**0. 395**

From Table 4, we see that NCP performs *significantly* better than CP with respect to 'discoverer', 'invention' and 'inventor', and it performs as well as CP with respect to 'invention year'.

<Discovery> was discovered by <Discoverer> in
<Discovery> {\w +9} was discovered by <Discoverer>.
<Discovery> {\w +8} discovered by <Discoverer> on
<Discovery> {\w +15} discovered by <Discoverer> in
<Discovery> {\w +10} discovered by <Discoverer> (
<Discovery> {\w +6} discovered in {\w +4} Sir <Discoverer>.
<Discovery> was discovered {\w +5} Sir <Discoverer>,

Figure 12: NCP Patterns with respect to Discoverer

Figure 12 shows the example patterns obtained by applying NCP to the 'discoverer' data.

6 Conclusion

This paper has presented a new algorithm for learning *non-consecutive* patterns *with constraints* for information extraction. The pattern generation step of the algorithm is an extension of the apriori algorithm.

We believe that the use of non-consecutive patterns is necessary, when it is to extract information *concerning with complex expressions* (note that flexibility or non-consecutiveness is the nature of language).

The main contributions of this paper, we think, are (a) the designing of the learning algorithm which can accurately and efficiently acquire non-consecutive patterns, (b) the proposal of using the constraints necessary for accurate and efficient information extraction, and (c) the empirical verification of the necessity of the use of non-consecutive patterns in information extraction.

References

H. Ahonen, O. Heinonen and M. Klemettinen, 1998. Applying Data Mining Techniques for Descriptive Phrase Extraction in Digital Document Collections. In *Proceedings of Advances in Digital Libraries Conference.*

R. Agrawal and R. Srikant, 1994. Mining Sequential Patterns. In *Proceedings of the 20th International Conference on Very Large Databases.*

S. Brin, 1998. Extracting Patterns and Relations from the World Wide Web. In *Proceedings of the WebDB Workshop at 6th International Conference on Extending Database Technology.*

S. Huffman, 1995. Learning Information Extraction Patterns from Examples. In *Proceedings of IJCAI-95 Workshop on New Approaches to Learning for Natural Language Processing*

B. Lent, R. Agrawal and R. Srikant, 1997. Discovering Trends in Text Databases. In *Proceedingsof the 3rd International Conference on Knowledge Discovery and Data Mining.*

D. Lin and P. Pantel. 2001. DIRT - Discovery of Inference Rules from Text. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001.*

H. Mannila and H. Toivonen, 1996. Discovering Generalized Episodes Using Minimal Occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining.*

D. Ravichandran and E. Hovy. Learning Surface Text Patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics.*

E. Riloff, 1996. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the 13th National Conference on Artificial Intelligence.*

S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert, 1995. Crystal: Inducing a Conceptual Dictionary. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

R. Srikant and R. Agrawal, 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology, EDBT.*

K. Sudo, S. Sekine and R. Grishman, 2001. Automatic Pattern Acquisition for Japanese Information Extraction. In *Proceedings of Human Language Technologies, 2001.*

E. Voorhees, 2001. Overview of the Question Answering Track. In *Proceedings of the TREC-10 Conference. NIST, Gaithersburg, MD, 157–165.*