IEICE TRANS. INF. & SYST., VOL.E90–D, NO.10 OCTOBER 2007

PAPER *Special Section on Knowledge, Information and Creativity Support System*

Automatic Extraction of the Fine Category of Person Named Entities from Text Corpora

Tri-Thanh NGUYEN[†a)], ***Student Member and*** **Akira SHIMAZU**[†b)], ***Member***

SUMMARY Named entities play an important role in many Natural Language Processing applications. Currently, most named entity recognition systems rely on a small set of general named entity (NE) types. Though some efforts have been proposed to expand the hierarchy of NE types, there are still a fixed number of NE types. In real applications, such as question answering or semantic search systems, users may be interested in more diverse specific NE types. This paper proposes a method to extract categories of person named entities from text documents. Based on Dual Iterative Pattern Relation Extraction method, we develop a more suitable model for solving our problem, and explore the generation of different pattern types. A method for validating whether a category is valid or not is proposed to improve the performance, and experiments on Wall Street Journal corpus give promising results.

key words: *fine person categories extraction, named entities, pattern extraction, algorithm*

1. Introduction

Named entities play important roles in many Natural Language Processing (NLP) applications [8]. The named entity (NE) set presented by the sixth Message Understanding Conference (MUC6) for application in business activities consists of 7 types: *organization*, *location*, *person*, *date*, *time*, *money* and *percent* [5]. Nonetheless, finer distinctions of NE are needed in some applications, thus Sekine proposed to extend the NE hierarchy to about 150 types (and currently about 200 types) [8], and Kiryakov presented a hierarchy of 250 NE types to support semantic search [6].

Though the NE sets of Sekine and Kiryakov contain relatively large numbers of types, current NE recognition systems usually assign a unique type to a named entity [3]. This approach does not reflect the real world, where a named entity can have more than one type. For example, a person NE can be both "*executive vice president*" and "*chief financial officer*". In addition, in the real application, such as question-answering (QA) systems, users may query the list of even finer categories of NEs, such as "*US presidents*". Fortunately, the actual fine category of a named entity may appear along with itself somewhere in the text, in certain patterns, as in the following example, where "*US president*" is the actual type of the named entity "*George Bush*":

George Bush, the US president, assembled . . .
He declared war against Iraq . . .

Given which, a user may ask "Which *US president* declared war against Iraq?" From the first sentence, if we could recognize the actual type of the named entity "*George Bush*", then from the second sentence with co-reference resolution, we can easily answer the above question. This is a possible application of finely categorized person named entities.

Pattern extraction was proposed and applied to solve many information extraction problems [1], [2]. Brin proposed an iterative model to extract (*author*, *title*) tuples that describe the relation: the author of the book *title* is *author* [2]. Based on Brin's model, Agichtein and Gravano presented the Snowball system for extracting (*organization*, *location*) tuples indicating that the headquarters of *organization* is in *location* [1].

This study proposes to extract the actual fine categories of person NEs by exploiting valuable hidden patterns in text documents in a data-driven way based on Brin's model. We start with seed patterns instead of seed tuples, so that the number of tuples extracted as well as the number of patterns generated in each iteration is consequently large, and the algorithm runs faster. We explore the generation of different pattern types for further extracting tuples, and propose a method for checking whether a newly extracted tuple (*person*, *category*) is valid or not; accordingly the precision is increased. The experiments with Wall Street Journal give relatively good results. A tuple (*person*, *category*) can be seen as the relation *person* is-a *category*, which is a useful relation and can be utilized in semantic search or QA systems, for example, for answering "Who is *person*?" questions.

The remainder of this paper is organized as follows: Section 2 summarizes some related work; Section 3 gives details about our algorithm; Section 4 presents experiments and evaluation, and the last section concludes the paper.

2. Related Work

Brin presented the "Dual Iterative Pattern Relation Extraction" (DIPRE) technique for extracting relations, and used DIPRE to extract (*author*, *title*) tuples having the relation: the author of the book *title* is *author*. Starting with a small number of (*author*, *title*) seed tuples, DIPRE finds the occurrences of tuples in order to generate new patterns. An occurrence of a (*author*, *title*) tuple is represented by a 7-tuple (*author*, *title*, *order*, *url*, *prefix*, *middle*, *suffix*), where

Manuscript received February 5, 2007.
Manuscript revised April 28, 2007.

[†]The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.
a) E-mail: t-thanh@jaist.ac.jp
b) E-mail: shimazu@jaist.ac.jp

DOI: 10.1093/ietisy/e90–d.10.1542

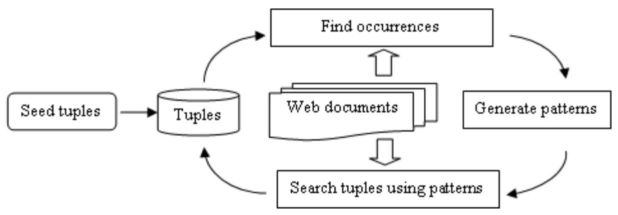

Fig. 1 Brin's DIPRE algorithm.

url is the URL of the document where the occurrence appeared; *order*, a boolean value, indicates the occurring order of the *author* and *title* in the text; if the *author* precedes the *title*, the *order* is true, otherwise it is false; *prefix* is *m* characters preceding the *author* (or *title* if the *title* is first); *middle* is the string between the *author* and *title*; *suffix* is *m* characters following the *title* (or *author* if the *author* is last). A new pattern (*order*, *urlprefix*, *prefix*, *middle*, *suffix*) is generated from a group of occurrences *O* having the same *url* (or prefix of *url*), if all the occurrences in *O* have the same *middle*, *order*, *prefix* (or some last characters of *prefix*) and *suffix* (or some first characters of *suffix*). New patterns are, then, used to extract further (*author*, *title*) tuples. The algorithm of DIPRE is depicted in Fig. 1.

Based on DIPRE, Agichtein and Gravano developed the Snowball system for extracting (*organization*, *location*) tuples expressing the relation: the headquarters of *organization* is in *location* [1].

Pasca presented a model based on DIPRE for acquiring (*C*, *N*) pairs (where *C* and *N* stand for category and named entity, respectively) from web documents by matching sentences with the template (pattern) [7]:

`[StartOfSent]` *C* `[such as|including]` *N* `[and|,|.]`

where *C* matches a plural noun; *N* matches consecutive proper nouns. Below are some sentences that match the above template, where *C* and *N* are underlined [7]:

- That is because software firewalls, including Zone Alarm, offer some semblance of this feature.
- API Adapter can be written in other programming languages such as C++.

Based on extracted pairs, new templates are generated to extract potential pairs. Pasca's model is interesting, because it is nearly an unsupervised approach. However, Pasca's model considers only categories that are expressed using plural nouns. In regular text documents, potential categories can be available using singular nouns. In addition, sentences that match this template do not appear frequently in all text corpora, e.g., the Wall Street Journal corpus which is used in our experiments. In this case, the approach may need a very large number of text documents in order to get a sufficient list of (*C*, *N*) pairs for further generation of new templates, as seen in the experiments of Pasca on a large dataset consisting of 500 million of web documents and news articles.

Our study can be seen as a complement to Pasca's study. We extract (*person*, *category*) tuples from text documents, in which categories are expressed using singular nouns. Though this study concentrates on person NEs, our model can be used to extract fine categories of other NE types, such as *organization* and *location*.

3. Extraction System

Among NE-related questions, ones concerning about person NEs comprise a relative large portion as seen in the question list in Text Retrieval Conference 9 QA track[†], so this study concentrates on extracting (*person*, *category*) tuples from plain-text corpora as a preliminary step, which can also be applied for extracting other types of (*NE*, *category*) tuples. Detailed descriptions of our model are given in the next subsections.

3.1 Extraction Algorithm

DIPRE starts with a small set of (*author*, *title*) seed tuples. The selection of these tuples must be done carefully, because if we select tuples that do not appear in the target corpus, then no pattern can be generated for extracting new tuples. If the seed tuples do not frequently occur, there may be a small number of new patterns discovered for further extraction, and the algorithm is time-consuming to scan the corpus several times for extracting new (*author*, *title*) tuples. In our model, we start with seed patterns as described in Sect. 3.2. By starting with seed patterns, the number of tuples extracted in the first scan is relatively large, consequently, the number of new patterns discovered for the next scan is large, and the algorithm may need fewer scans on the corpus.

A new (*author*, *title*) tuple is extracted if there is a document with URL matching *urlprefix*∗, and the document contains texts matching the regular expression:

∗*prefix*, *author*, *middle*, *title*, *suffix*∗

which is constructed from the pattern (*order*, *urlprefix*, *prefix*, *middle*, *suffix*), where the *order* (indicating the order of *author* and *title*) is true. Non-empty strings for *prefix*, *middle* and *suffix* are used to determine the boundary of *author* and *title*. If we use DIPRE to extract (*person*, *category*) tuples, this regular expression fails to extract (*person*, *category*) tuples whose *person* appears at the beginning (or the end) of a sentence. Moreover, all *prefix*, *middle* and *suffix* should not be a space, since a space does not specify a clear boundary of *person* and *category*. Thus, the pattern fails to work in situations where *person* and *category* are separated by a space, such as "*He demanded that Treasury Secretary Nicholas Brady appear before the Senate Banking Committee to explain...*"[††], where the *category* is "*Treasury Secretary*", and the *person* is "*Nicholas Brady*".

To avoid the above problems, we use a named entity recognition (NER) system to determine the boundary of person NEs. Because the *category* of a (*person*, *category*) tuple

[†]http://tangra.si.umich.edu/clair/NSIR/cgi-bin/trec-question.cgi?collection=9&script=html/nsir.cgi
[††]An example taken from the Wall Street Journal corpus.

```
Input: A seed pattern set $P_1$; a text corpus $D$;
Output: The list $L$ of (person, category) tuples;

  1. Initiation: the pattern set $P \leftarrow P1$; $L \leftarrow \emptyset$ (empty list)
     Find named entities in every sentence in $D$;
     $D \leftarrow D$ - {sentence | sentence contains no person NE}
     Add part-of-speech and chunks tags for every sentence in $D$;
  2. Extract the list $L'$ of (person, category) tuples from
     sentences that match any pattern in $P$; $L \leftarrow L + L'$;
     Let $D'$ be the list of sentences, from which the
     (person, category) tuples in the list $L'$ were extracted;
     $D \leftarrow D$ - $D'$;
     If $D$ is empty, then return;
  3. Find the list $O$ of the occurrences of (person, category) tuples in $D$;
  4. From the list of occurrences $O$, generate a new pattern
     set $P'$; $P \leftarrow P'$;
     If $P$ is empty, then return, otherwise go to Step 2;
```

Fig. 2 (*person*, *category*) tuples extraction algorithm.

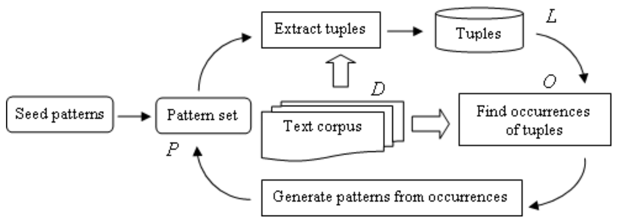
Extract tuples
Tuples
L
D
O
Seed patterns
Pattern set
Text corpus
Find occurrences of tuples
P
Generate patterns from occurrences

Fig. 3 Our (*person*, *category*) extraction model.

is a noun phrase, we use a shallow parser to identify the boundary of *category*. Thus, our patterns do not need *prefix* and *suffix* components, and the method for generating new patterns is different from Brin's as discussed in Sect. 3.3. For improving the performance of the algorithm, we propose to use a function to validate a *category* in the extraction process as discussed in Sect. 3.4. Our algorithm is described in Fig. 2.

In Step 2 of our algorithm, we remove sentences from which one or more tuples were extracted, because if we keep them for later scans, then duplicate tuples can be extracted. In Step 4, we replace *P* by *P′* (the set of newly generated patterns), since the old pattern set *P* in the previous scan cannot extract new tuples in the next scan. The graphical demonstration of our algorithm is depicted in Fig. 3.

3.2 Clue and Seed Patterns

Our seed patterns are based on *appositives*. Grammatically, an appositive is a noun phrase that renames or describes another noun phrase, with no word interposed between the two phrases. For example, in the sentence "*George Bush, the US president, announced* ···", the appositive "*the US president*" describes more concretely the person named entity "*George Bush*", and "*US president*" can be regarded as the actual type of "George Bush". In another complex example of appositive: "*Daniel Akerson, executive vice president and chief financial officer, said MCI's growth is being fueled by* ···"[†], two noun phrases: "*executive vice president*" and "*chief financial officer*" describe "*Daniel Akerson*", so two tuples are expected to be extracted. We only consider appositives whose head is a singular noun (tagged NN or NNP[††]), because a noun that describes a person should be in singular form.

Because we work on sentences which have been parsed by a shallow parser, our patterns may contain part-of-speech (POS) and chunk tags. A chunk is a syntactically related non-overlapping group of words. A chunk is assigned a tag, such as NP (noun phrase), and surrounded by a square bracket pair, e.g., "[NP executive/JJ vice/NN president/NN]".

Pattern: We define a *pattern* as a 4-tuple:

(*order*, *person_slot*, *middle*, *category_pattern*),

where *order* indicates the occurrence order of *person* and *category* in a sentence. If *person* is before *category*, the *order* is NE then *category* (hereafter we call this NEC for short), otherwise *order* is *category* then NE (hereafter we call this CNE for short). *person_slot* is a slot which will be replaced with a person NE (with POS tags) that appears in the sentence currently being processed. For example, if the current sentence has the person named entity "*George Bush*", then *person_slot* is: George/NNP Bush/NNP. Let *simple_noun* be a pattern matching a noun phrase with POS tags which consists of one or zero determiner, adjectives, gerunds and nouns, e.g., "a/DT managing/VBG director/NN":

```
simple_noun:=($word/DT )?($word/(JJ|VBG) )*
        ($word/NNP?S? )*($word/NNP? )+[†††]
```

Then *category_pattern* is defined as:

```
category_pattern:=
        simple_noun1 (and/CC simple_noun2)?[††††]
```

The sentence "*George Bush, the US president, announced* ···", after having been parsed, has the form: [NP George/NNP Bush/NNP] ,/, [NP the/DT US/NNP president/NN] . . . , so the *middle* component for the first seed pattern is: "] ,/, [NP ", and the first seed pattern is:

(NEC, *person_slot*, "] ,/, [NP " , *category_pattern*)

A person NE may also lie in an appositive, e.g., "*Semi-Tech's president and chief executive officer, James Ting, said*

[†]An example taken from the Wall Street Journal corpus.

[††]NN and NNP are used to tag singular lowercase and singular proper nouns, respectively.

[†††]? stands for "there is zero or one"; | stands for "or"; + stands for "there is one or more"; * stands for "there is zero or more."

[††††]This pattern covers the cases where there are two noun phrases in the appositive.

it was likely that the Singer board would approve ...". In this example, the appositive "*James Ting*" renames "*president and chief executive officer*". Thus, the second seed pattern is:

(CNE, *person_slot*, "] ,/, [NP " , *category_pattern*).

If the *order* is NEC, then from a pattern:

(*order*, *person_slot*, *middle*, *category_pattern*),

where *person_slot* is replaced with a *person* NE (with POS tags) that appeared in a sentence s, the regular expression:

∗*person_slot*, *middle*, *category_pattern*∗

is constructed to match s. If the *order* is CNE, then *person_slot* and *category_pattern* are reversed.

Let *person* be a *person_slot* after removing POS tags. Let *category* be a *simple_noun* after removing the possible determiner (tagged DT) and POS tags. If a match is found, the expected tuples are (*person*, $category_1$) and possible (*person*, $category_2$). This is an advance from DIPRE, because our method can extract two tuples from a match if there are two.

3.3 Pattern Generation

Similar to Brin's model, our extraction model exploits the fact that (*person*, *category*) tuples can be expressed in different lexical forms, which tend to appear in uniform patterns repeated in collections of documents. For example, the tuple ("*George Bush*", "*US president*") can be expressed in different ways as follows:

George Bush, the US president, announced ...

US President George Bush announced ...

Occurrence: Similarly to Brin's model, we define an occurrence of a (*person*, *category*) tuple as a 4-tuple:

(*order*, *person*, *middle*, *category*)

where *order* has the same meaning as that of our patterns; *middle* is the string surrounded by *person* and *category*. Based on the list of occurrences, we explore the method for generating different pattern types in the next subsections.

3.3.1 Exact Patterns

Occurrences of *person* and *category* are used to generate new patterns. However, a *middle* of an occurrence is not necessarily reliable, we need a method to retain reliable ones. Our method is based on two criteria: *repetition* and *diversity* as follows:

Repetition of a *middle* (repetition(*middle*)) is the number of times the *middle* appears between the *person* and *category* of (*person*, *category*) tuples which have the same *person*.

Diversity of a *middle* (diversity(*middle*)) is the number of times the *middle* appears between the *person* and *category* of (*person*, *category*) tuples which have different *persons*.

A *middle* that has repetition(*middle*) > $threshold_R$ seems reliable and is kept. A pattern seems specific if it is generated based on tuples of a *person*, so we only keep *middles* that have diversity(*middle*) > $threshold_D$ to make the generated patterns general (Condition 1).

If a *middle* contains a verb phrase, the verb phrase should express the relation *person* is-a *category* (Condition 2).

Some valid verbs are: '*be*', '*assign*', '*elected*', '*take over*', '*name*', '*continue*', '*remain*'. We also care about the tense of these verbs. For some verbs, such as '*be*', '*assign*', and '*elect*', we do not accept their future or future perfect tenses. Because, for example, "a *person* will be a *category*" does not certainly mean the *person* is a *category*. The *middle* that contains a verb phrase is retained if it satisfies this constraint.

These two conditions are used in the pattern generation procedure described in Fig. 4. We call these patterns *exact patterns*.

```
1. Group all occurrences in the list $O$ by $order$ and $middle$; Let the resulting groups be $O_1, O_2, \dots, O_N$;
2. For each group $O_i$, if the $middle$ satisfies the two conditions, generate a new pattern:
       $(order, person\_slot, middle, category\_pattern)$;
```

Fig. 4 Pattern generation procedure.

Table 1 Examples of the *middle* of exact patterns.

Middle and matched sentences	Order
Middle:] ,/, [NP Apple/NNP] [NP 's/POS Sentence: ... [VP says/VBZ] [NP Randall/NNP Battat/NNP] ,/, [NP Apple/NNP] [NP 's/POS *product-marketing*/JJ *vice*/NN *president*/NN] ./.	NEC
Middle: A space Sentence: [PP that/IN] [NP *Treasury*/NNP *Secretary*/NNP Nicholas/NNP Brady/NNP] [VP appear/VBP] [PP before/IN] [NP the/DT Senate/NNP Banking/NNP Committee/NNP] [VP to/TO explain/VB] ...	CNE

Examples of *middle* and sentences that match the corresponding exact patterns are given in Table 1, where Order is the *order* of the exact patterns; *person* NEs are underlined and *categories* are in italics.

Exact patterns are relatively reliable; however, they have narrow coverage, so we propose other ways for extending coverage, as given in the next subsections.

3.3.2 Sketch Patterns

For the *middle* of an exact pattern having the *order* NEC:

"] ,/, [NP ABC/NNP] [NP 's/POS" (1)

Table 2 Examples of the *middle* of sketch patterns.

Middle and matched sentences	Order
Middle:] ,/, [NP who/WP] [VP is/VBZ] [NP $*word*/NNP] [NP 's/POS Sentence: [NP Mr./NNP Petit/NNP] ,/, [NP who/WP] [VP is/VBZ] [NP Healthdyne/NNP] [NP 's/POS *chairman*/NN . . .	NEC
Middle:] [PP of/IN] [NP $*word*/DT $*word*/NNP $*word*/NNP $*word*/NNP] ,/, [NP $*word*/NNP Sentence: [NP A/DT *former*/JJ *governor*/NN] [PP of/IN] [NP the/DT Spanish/NNP Central/NNP Bank/NNP] ,/, [NP Mr./NNP Rendueles/NNP] . . .	CNE

its corresponding exact pattern can match the sentence:

[NP Harvey/NNP Dzodin/NNP] ,/, [NP ABC/NNP] [NP 's/POS vice/NN president/NN] . . . (2)

However, this pattern cannot match a similar sentence that describes the "*director*" of another company (organization), e.g., IBM, in the same syntax as (2):

[NP Alan/NNP Baratz/NNP] ,/, [NP IBM/NNP] [NP 's/POS director/NN]. . . (3)

If we modify the *middle* (1) so that its pattern can match (3), then expected tuples in both (2) and (3) can be extracted. In order to do this, we convert (1) into a template that can match other sequences having similar structure, except for nouns, adjectives, cardinals or articles. Concretely, we replace nouns, adjectives, cardinals and articles in a *middle* with a variable $*word* that matches a word. Below is the template constructed from the *middle* (1):

"] ,/, [NP $*word*/NNP] [NP 's/POS " (4)

We keep other words in a *middle* intact, such as verbs, prepositions or conjunctions, because their modification can make the context of the *middle* different. We call this template the *sketch* of a *middle*. We produce a new pattern type that we call *sketch patterns*, of which the *middle* component is replaced with a *sketch*.

Examples of *middles* of sketch patterns and matched sentences are given in Table 2.

3.3.3 Extended Sketch Patterns

Let's consider the sketch (*middle* component) of a sketch pattern with *order* NEC:

"] ,/, [NP $*word*/NNP $*word*/NNP] [NP 's/POS " (5)

The pattern can match the sentence:

[NP Bill/NNP Gates/NNP] ,/, [NP Microsoft/NNP Corporation/NNP] [NP 's/POS chairman/NN] . . . (6)

Table 3 The *middle* of an extended sketch pattern.

Middle and matched sentence	Order
Middle:] ,/, [NP $*word*/DT($*word*/NNP)+] ,/, [NP $*word*/NNP] ,/, [NP Sentence: [NP James/NNP Bopp/NNP Jr./NNP] ,/, [NP the/DT Terre/NNP Haute/NNP] ,/, [NP Ind./NNP] ,/, [NP *lawyer*/NN] [NP who/WP] [VP filed/VBD] [NP the/DT high-court/NN . . .	NEC

However, this pattern can not match the sentence:

[NP Alfonso/NNP J./NNP Fanjul/NNP Jr./NNP] ,/, [NP Southeast/NNP Banking/NNP Corp./NNP] [NP 's/POS director/NN] . . . (7)

If the sketch (5) could be extended so that its corresponding pattern matches (7), then expected tuples in (6) and (7) would be extracted. We do this by generalizing the noun phrase of sketch (5) to enable it to match noun phrases which have one or more proper nouns. Concretely, we replace *consecutive* (proper) nouns (or adjectives) template in a sketch with another template that can match one or more consecutive (proper) nouns (or adjectives). For example, (5) is generalized as:

"] ,/, [NP($*word*/NNP)+] [NP 's/POS " (8)

We call a generalized sketch an *extended sketch*, and introduce a new pattern type called *extended sketch pattern*, of which the *middle* component is replaced with an *extended sketch*.

Table 3 gives an example of the *middle* of an extended sketch pattern and a sentence that matches the pattern.

3.3.4 Pattern Generation Order

Obviously, the tuples extracted by exact patterns are a subset of those extracted by sketch patterns; the tuples extracted by sketch patterns are a subset of those extracted by extended sketch patterns. Thus, we give exact patterns highest priority and extended sketch pattern the lowest priority, and run lower priority (or larger coverage) patterns only on the remaining dataset, which is obtained after processing by higher priority patterns.

3.4 Category Validation

Though our algorithm runs on documents, in which person named entities have been tagged by an NER system, the NER system may incorrectly assign person type to a proper name which is actually of another type, e.g., an *organization*. Consequently, the system may extract an incorrect tuple. Also, not all new patterns are 100% reliable, so some extracted (*person*, *category*) tuples are incorrect, and should be discarded.

We propose an additional method for validating

```
for every sentence $s$ in $D$ do
  for every $person$ NE in $s$ do
    for every pattern $p$ in $P$ do
      Construct a regular expression $r$ from $person$ and $p$;
      if ($r$ matches $s$)
        Extract possible ($person$, $category$) tuples;
        if ($category$ is valid) return the tuples ;
        //Skip other patterns
        else Discard the tuples;
    end for
  end for
end for
```

Fig. 5 Tuples extraction with the validation function.

whether a newly extracted tuple (*person*, *category*) is correct or not. If a *person* is a *category*, then the head noun of the noun phrase describing *category* must be a sort of person. In other words, the *category* must be a subtype (more specific type) of person. The subtype relation is represented as *hyponym* relation in WordNet [4]. Thus, a *category* is valid if it is a hyponym of person. The reverse relation of hyponym is *hypernym*, so it is equivalent to say a *category* is valid if its hypernym is a person. Checking whether person is a hypernym of a *category* seems faster than checking whether a *category* is a hyponym of person, because the list of hyponyms of person is relatively large. We used WordNet to check hypernym relations for validation.

A word may have more than one *sense* (or meaning), and there may be multiple hypernyms for each sense. If all senses of a word are checked whether their hypernyms are persons or not, we may get unexpected results. For example, if all senses of the words: '*study*', '*guide*' or '*computer*' are checked, then one of their hypernyms is a person. For this reason, only the first sense of a given word is checked whether its hypernyms be a person or not. This constraint helps to improve the precision of the algorithm; however, it has a negative effect to the recall, since it fails to validate words (e.g., '*justice*') in case a sense other than its first sense is used. Fortunately, its negative effect is little.

The validation function is integrated in Step 2 of the algorithm in Fig. 2, and is described in Fig. 5. From each sentence, when the validation function is not used, the algorithm extracts tuples only at the first match, which is not always the expected match.

4. Experiments and Evaluation

4.1 Text Dataset

We used the Wall Street Journal (WSJ) corpus, which consists of 595 files, as the dataset. After extracting the body part and removing other parts, e.g., the headers, we got a plain text collection with the size of 308 MB consisting of nearly 3 million sentences. In the initiation step of the algorithm in Fig. 2, we tagged all named entities in this plain text collection by an NER system. Through an investigation over free open source NER systems, we considered two competitive systems: OpenNLP[†] and LingPipe[††]. For NER task, OpenNLP is rather slow in comparison with LingPipe, so we used LingPipe for tagging named entities. After removing sentences that contained no person NE, 667,981 sentences were collected. Next step was to add POS and chunk tags for each sentence. In this task, OpenNLP was used because it has better performance than LingPipe.

Table 4 Results of baseline and the PCE with threshold of 3.

Pattern	PCE-baseline			PCE 3		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
Seed	89.03	35.84	51.11	99.26	35.06	**51.82**
Exact	63.41	72.47	67.64	94.48	75.58	**83.98**
Sketch	62.88	74.81	68.33	94.50	80.26	**86.80**
E. sketch	62.88	74.81	68.33	94.50	80.26	**86.80**

We ran our programs on 667,981 sentences to get patterns, and tested the precision and recall of the patterns on a test set. The test set is 1,000 sentences which are randomly selected from the WSJ corpus. From the test set, 385 (*person*, *category*) tuples were manually extracted.

4.2 Experiments

We used C++ with the regular expression library *boost*[†††] to build the programs. Let PCE (*Person-Category Extraction*) be the algorithm with the category validation function. We also wrote a similar program called "PCE-baseline" which has no category validation function. We calculated precision (P), recall (R) and F-score (F) as follows:

$$P = \frac{T_{correct}}{T_{extract}} \times 100\%, R = \frac{T_{correct}}{385} \times 100\%, F = \frac{2RP}{R+P}$$

where $T_{correct}$ was the number of tuples correctly extracted; $T_{extract}$ was the number of tuples extracted by the program.

In general, the repetition threshold $threshold_R$ can be different from the diversity threshold $threshold_D$, so there are many combinations of the values of these variables. In our experiments, for the sake of simplicity, we set $threshold_R$ and $threshold_D$ to the same value. Later, we simply called them *threshold* for short. Table 4 shows the results of different pattern types (*seed*, *exact*, *sketch* and *extended sketch* patterns) of the PCE-baseline and PCE with the threshold of 3. This threshold is based on trying several values. From the results, we can see the important effect of validation function to the performance.

In order to investigate a proper threshold, we ran PCE with the thresholds of 4 and 5. Results in Table 5 show that 3 seems to be the proper value of the threshold. Figure 6 shows the number of accumulated tuples extracted by PCE with different thresholds and pattern types from the dataset of 667,981 sentences. Figure 6 also shows that the number of tuples extracted by seed patterns is about 41.6% (a relatively large portion) of the total number of extracted tuples. Especially, the precision of the seed patterns is very high

[†]http://opennlp.sourceforge.net/

[††]http://www.alias-i.com/lingpipe/index.html

[†††]http://www.boost.org

Table 5 Results of the PCE with threshold of 4 and 5.

Pattern	PCE 4			PCE 5		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
Seed	99.26	35.06	51.82	99.26	35.06	51.82
Exact	94.46	75.32	83.82	94.46	75.32	83.82
Sketch	94.48	80.00	86.64	94.46	79.74	86.48
E. sketch	94.48	80.00	86.64	94.46	79.74	86.48

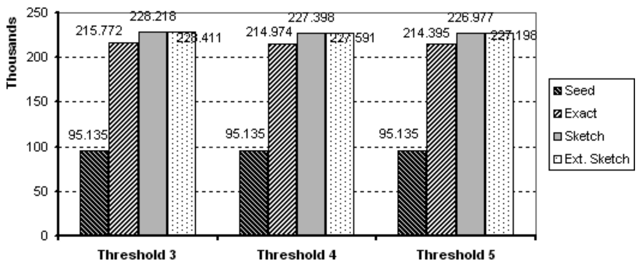

Fig. 6 Extracted tuples at different thresholds.

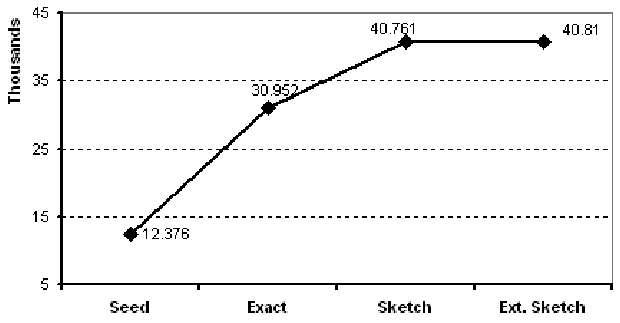

Fig. 7 The growth of the number of distinct categories.

Table 6 Some top and bottom categories with frequency.

President (22679), Chairman (12835), Analyst (6729), Vice President (6011), Director (5821), Chief Executive Officer (5326), Judge (5050), Dr. (4931), Rep. (3479), Senior Vice President (3028), Executive Vice President (2698), Attorney (2537), Managing Director (1904), Chief Executive (1834), Chief Economist (1706), Lawyer (1601), Manager (1586), Economist (1510), Editor (1477)
part-time CIA employee (1), partnership analyst (1), parliament deputy (1), parts marketing administrator (1), past finance director (1), patent specialist (1), paintings specialist (1), personal translator (1), freight carrier (1)

(99.26%) proving that the selection of seed patterns is good.

Figure 7 shows the growth of distinct categories of PCE with threshold of 3 from the dataset of 667,981 sentences. The figure shows that the number of potential categories is relatively large (it reaches 40,810 for extended sketch patterns).

Table 6 lists some top and bottom ranked categories along with their frequency that are extracted by our system.

Excluding the processing time of the Initiation step of the algorithm in Fig. 2, our programs take about 40 hours.

We also implemented another version of PCE called RPCE (*Reverse* PCE) which starts with the seed tuples instead of seed patterns. A program similar to RPCE without the category validation function was coded as the corresponding baseline which we called RPCE-baseline. The seed set consisted of 20 (*person*, *category*) tuples that are randomly selected from the WSJ corpus. The thresholds of both RPCE and RPCE-baseline are 3. And the results of the experiment on the same dataset are given in Table 7.

Table 7 Results of the RPCE with threshold of 3.

Pattern	RPCE-baseline			RPCE 3		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
Exact	61.70	69.87	65.53	94.28	72.73	82.11
Sketch	61.10	72.21	66.19	94.30	77.40	85.02
E. sketch	61.10	72.21	66.19	94.30	77.40	85.02

Table 8 Number of generated middles.

	Exact		Sketch		E.Sketch		Total
	NEC	CNE	NEC	CNE	NEC	CNE	
PCE 3	198	57	84	31	15	0	385
PCE 4	150	46	73	25	12	0	306
PCE 5	118	44	68	22	12	0	264
RPCE 3	196	54	79	32	8	0	369

Except for the first scan, which took about 35 minutes, for generating new patterns from seed (*person*, *category*) tuples, the computation time of RPCE was about the same as that of PCE.

Table 8 records the number of middles generated by our programs, and this is also the number of patterns. Some NEC middles of exact patterns that are detected by PCE-3 but are not detected by RPCE-3 are listed below:

```
" ] ,/, [NP Falconbridge/NNP ] [NP 's/POS"
" ] ,/, [NP Fannie/NNP Mae/NNP ] [NP 's/POS"
```

From our observations, there are some factors that decrease the recall of PCE, as follows:

- There are some sequences which do not satisfy the criteria for generating a new pattern.
- LingPipe incorrectly assigned some named entities as person NEs, while they were actually of another type, such as *organization*.
- A person may be represented by a pronoun: "*He is a worker*", where the pronoun '*He*' refers to a person NE in a preceding sentence. This sentence is ignored because there is no person NE.
- Person NEs that are not recognized by LingPipe can not be extracted, because the sentences containing them are removed in Initiation step.
- The category validation function fails to validate a *category* whose head noun has a meaning different from its first sense.
- Category may be expressed in plural form, such as "*John is one of the well-known leaders*". We do not treat such cases.
- A sentence may contain several tuples, however not all of them are extracted because the pattern set in a certain iteration does not cover all tuples of that sentence, so

the sentence was removed in Step 2 of the algorithm in Fig. 2.

Among the above reasons, the first reason is most often responsible, in comparison with the rest.

The main reason that decreases precision is the generation of incorrect patterns.

5. Conclusion

In this paper, we proposed a method for extracting actually categories of person named entities from text documents in a data-driven way. We proposed new constraints for generating new patterns, as well as a method for validating whether a new tuple is correct or not. We performed experiments on the Wall Street Journal corpus, and obtained relatively good results.

In the current implementation, we have not taken the priority of patterns in the same pattern set, so that more reliable pattern is selected before less reliable ones with the purpose to improve the precision.

Our model can be applied to extract fine categories of other named entity types, such as organization and location.

The relation is-a in a tuple (*person*, *category*) is valuable, so its application in a semantic search system is an interesting direction in future work.

Acknowledgements

This study was supported by Japan Advanced Institute of Science and Technology, the 21st Century COE Program: “Verifiable and Evolvable e-Society”.

References

[1] E. Agichtein and L. Gravano, “Snowball: Extracting relations from large plaintext collections,” Proc. 5th ACM International Conference on Digital Libraries, pp.85–94, San Antonio, 2000.
[2] S. Brin, “Extracting patterns and relations from the World Wide Web,” Proc. 6th International Conference on Extending Database Technology, pp.172–183, Springer, Valencia, 1998.
[3] H. Chieu and N. Tou, “Named entity recognition with a maximum entropy approach,” Proc. CoNLL-2003, pp.160–163, Edmonton, 2003.
[4] C. Fellbaum, ed., WordNet: An electronic lexical database and some of its applications, MIT Press, 1998.
[5] R. Grishman and B. Sundheim, “Message understanding conference 6: A brief history,” Proc. COLING-96, pp.466–471, 1996.
[6] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff, “Semantic annotation, indexing, and retrieval,” Elsevier’s J. Web Semantics, vol.2, no.1, pp.49–79, 2004.
[7] M. Pasca, “Acquisition of categorized named entities for Web search,” Proc. 13th ACM Conference on Information and Knowledge Management (CIKM-04), pp.137–145, Washington, 2004.
[8] S. Sekine, K. Sudo, and C. Nobata, “Extended named entity hierarchy,” Proc. Third International Conference on Language Resources and Evaluation, pp.1818–1824, Canary Island, 2000.

Tri-Thanh Nguyen received the Bachelor degree in Informative Technology Faculty, Vietnam National University of Hanoi (VNUH) in 1999, and MSc degree in Asian Institute of Technology (AIT), Thailand in 2002. From February 2003, he worked as a lecturer for Faculty of Information Technology, School of Technology, VNUH. Since April 2005, he has been a PhD student in Natural Language Processing Laboratory, School of Information Science, Japan Advanced Institute of Science and Technology (JAIST).

Akira Shimazu is a professor in the Graduate School of Information Science at Japan Advanced Institute of Science and Technology from 1997. He received a Bachelor and Master degrees in mathematics from Kyushu University in 1971 and 1973, respectively, and a Doctoral degree in natural language processing from Kyushu University in 1991. From 1973 to 1997, he worked for NTT Laboratories. From 2002 to 2004, he was the president of the Association for Natural Language Processing.