A Pattern-Based Approach to Recognizing Time Expressions

Wentao Ding, Guanji Gao, Linfeng Shi, Yuzhong Qu
National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing, China
{wtding, gjgao, lfshi}@smail.nju.edu.cn, yzqu@nju.edu.cn

Abstract

Recognizing time expressions is a fundamental and important task in many applications of natural language understanding, such as reading comprehension and question answering. Several newest state-of-the-art approaches have achieved good performance on recognizing time expressions. These approaches are black-boxed or based on heuristic rules, which leads to the difficulty in understanding the temporal information. On the contrary, classic rule-based or semantic parsing approaches can capture rich structural information, but their performances on recognition are not so good. In this paper, we propose a pattern-based approach, called PTime, which automatically generates and selects patterns for recognizing time expressions. In this approach, time expressions in training text are abstracted into type sequences by using fine-grained token types, thus the problem is transformed to select an appropriate subset of the sequential patterns. We use the Extended Budgeted Maximum Coverage (EBMC) model to optimize the pattern selection. The main idea is to maximize the correct token sequences matched by the selected patterns while the number of the mistakes should be limited by an adjustable budget. The interpretability of patterns and the adjustability of permitted number of mistakes make PTime a very promising approach for many applications. Experimental results show that PTime achieves a very competitive performance as compared with existing state-of-the-art approaches.

1 Introduction

Along with the rapid progress of natural language understanding research area, understanding temporal information has been increasingly important in various applications such as reading comprehension and question answering. Recognizing time expressions, as a first and fundamental step of understanding temporal information, has attracted considerable attention since last decade. Lots of efforts have been paid on developing standards for modeling temporal information and annotating time expressions in free text (Pustejovsky et al. 2005; 2010; Hobbs and Pan 2006). Many annotated data corpus, such as the datasets of TempEval (Verhagen et al. 2010; UzZaman et al. 2013), the WikiWars dataset (Mazur and Dale 2010; Lee et al. 2014) and the Tweets dataset (Zhong, Sun, and Cambria 2017) , were built for evaluating the ability of time expression recognizing systems.

Existing approaches for time expression recognizing have achieved good performance on benchmark datasets, especially two recent approaches named SynTime and TOMN. SynTime (Zhong, Sun, and Cambria 2017) improved previous results by grouping tokens to 3 main types, including *time token*, *modifier* and *numeral*. It uses the time tokens to trigger time expressions and expand their boundaries by searching modifiers and numeral tokens with heuristic rules. TOMN (Zhong and Cambria 2018) use the three types as pre-tags instead of the classic BIO tagging scheme with a CRF model to achieve state-of-the-art results.

However, TOMN is a black-boxed learning approach, thus its result lacks an interpretability, and has no structure information for further understanding. SynTime expands the boundary of time expression by heuristically searching tokens of "PREFIX", "SUFFIX" or *numeral*. Though having a good performance, the heuristics also reduced the interpretability of SynTime and limited its adaptivity. For example, the word "in" can appear within a time expression like "⟨*T*⟩*late* <u>*in*</u> *2018*⟨*/T*⟩" or in front of a time expression like "*finished* <u>*in*</u> ⟨*T*⟩*2017*⟨*T*⟩", which cannot be distinguished by the heuristics. On the contrary, the classic rule-based or semantic parsing approaches can capture rich structural information, but their performances have a clear gap comparing with those newest approaches. Besides, different applications may have different precision-recall requirements on recognizing time expressions, which is not seriously considered in the existing approaches.

The previous work has shown the power of token types in recognizing time expressions. With these types, it is possible to generate sequential patterns from known time expressions. The generated patterns can be used to recognize new time expressions from incoming texts, and their explicit sequential structures can also give an interpretability to the recognized results. However, since the actual natural language utterances can be highly ambiguous and complex, the generality of token types can also bring mistakes. For example, as shown in figure 1, the pattern "NUMBER TIME_UNIT" matches three different time expressions, "29 years", "two days" and "one month", but it mistakenly matches a non-timex string "three quarters", in which the token "quarter" means "1 of 4 equal parts" but not 15 minutes or 3 months.

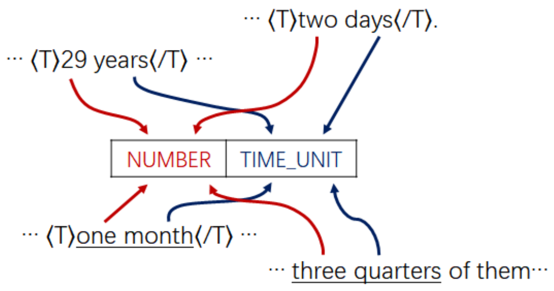

Figure 1: An example of pattern and its instances

This brings an interesting question: is it possible to select an appropriate subset of all generated patterns, to achieve a good performance on recognizing time expressions, and meanwhile provides an adjustability on limiting the total mistakes for fitting different precision-recall demands of various applications?

In this paper, we proposed a pattern-based approach to recognizing time expression, PTime. Like many existing rule-based approaches, we use regex patterns to detect and type time-related tokens. We automatically generate candidate patterns from the training time expressions, and then select an appropriate subset of them to maximize the correct token strings matched by the selected pattern set while the number of total mistakes caused by it is limited. Therefore, the pattern selection problem should be modeled as an optimization on the whole set of candidate patterns. We model the pattern selection problem as the Extended Budgeted Maximum Coverage (EBMC) Problem (Ding et al. 2015), which is to optimize a coverage function with a linear constraint on the cost of selecting an object. As the EBMC model has an explicit budget to constrain the total cost, a parameter ρ is introduced to loosely bound the total mistakes caused by the selected patterns. A lower ρ allows more mistakes, which may enlarge their coverage while decreasing the precision, and the result is just the opposite for a higher ρ.

The rest of this paper is structured as follows. Section 2 reviews related work. In section 3, we present the frame-work of PTime. In section 4, we describe the details in pattern generation. In section 5, we propose our EBMC-based pattern selection method and a greedy algorithm to implement the EBMC-based selection method. In section 6, we report our experiment results on three benchmark datasets. Finally, the last section concludes the paper with future work.

2 Related Work

The processing of temporal information can be divided into two subtasks, recognition and normalization of time expressions. We focus on the first subtask in this paper. Existing approaches for time expression recognition can be roughly classified as rule-based and learning-based approaches, and semantic parsing approaches can be regarded as a hybrid of them.

Classic rule-based approaches use deterministic hand-engineered rules to recognize and type time-related tokens and strings. Earlier work has shown that the complexity of time expression is bounded, thus finite state automata can be effective for extracting them from text (Hobbs et al. 1997). TempEX (Mani and Wilson 2000), and its expansion, GUTime (Verhagen et al. 2005), use both hand-crafted rules and machine-learnt rules for extracting temporal information from text. HeidelTime (Strötgen and Gertz 2010; Strötgen, Zell, and Gertz 2013; Strötgen et al. 2014) manually designs rules for recognizing time tokens and modifiers, combining tokens, and filtering ambiguous expressions. SUTime (Chang and Manning 2012) proposed a 3-layered temporal pattern language. It firstly recognizes single tokens, then expands tokens to strings, and finally composes and filters the strings to get time expressions.

Semantic parsing approaches relies on hand-engineered composition grammars defined on syntactic or semantic units. (Angeli and Uszkoreit 2013) uses an EM-style bootstrapping approach to learn a PCFG parser on pre-defined preterminals. UWTime (Lee et al. 2014), uses a combinatory categorical grammar to construct a compositional meaning representations, and trains a context-dependent semantic parser for parsing temporal information.

The above work recognize time expressions with rich structural information, while the recall of their results are unsatisfied. A recent work, SynTime (Zhong, Sun, and Cambria 2017), focus on recognition and impressively outperforms other rule-based approaches. Instead of designing rules in a fixed method, SynTime uses a group of time-related tokens types to trigger time expressions and expands their boundaries by several generic but heuristic rules.

The state-of-the-art black-box learning approaches use classic sequential tagging models (e.g. CRF, Maximum-Entropy Markov model) for the recognition problem. They characterize a token by extracted morphologic or syntactic features and temporal types. To capture the structure information coarsely, they tag a token by both its own features and the features of its preceding and following words. ClearTK-TimeML (Bethard 2013) uses a small set of morpho-syntactic features and temporal types of alphanumeric sub-tokens. TOMN (Zhong and Cambria 2018) uses the same token regex expressions as SynTime to recognize *time token*, *modifier* and *numeral* as pre-tags. The new tagging scheme outperforms other classic schemes like the BIO scheme with POS-tags as features.

Expert knowledge make most of the classic rule-based approaches and the semantic parsing approaches have interpretability, which could be helpful for understanding task like time expressions classification and normalization. Meanwhile, existing approaches with good performance are black-boxed or based on heuristic rules. Our approach also uses token types, but it automatically generates candidate patterns and then selects appropriate subset of generated patterns based on the EBMC model. Besides, a parameter is introduced to balance the precision-recall demand.

3 The Framework of PTime

In this section, we explain the framework of PTime. As indicated by figure 2. Our approach consists a training process and a test process.

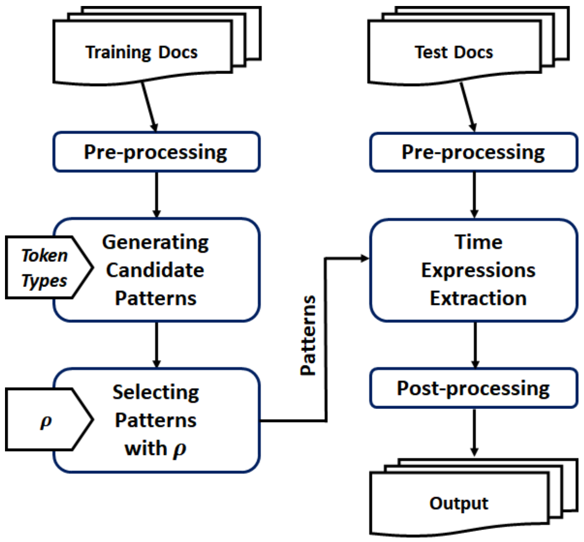

Figure 2: The framework of PTime

3.1 The Training process

In the training process, PTime transforms the input documents to token sequences during pre-processing, generating patterns based on predefined token types, then select an appropriate subset of the candidates according to the precision-control parameter ρ.

Pre-processing As the first step of our approach, the pre-processing component transforms the input documents to token sequences, we use CoreNLP to get the lemma and the POS-tag of each token. Most of the lemma-POS pairs will be regarded as tokens directly. However, some of the tokens are written as combinations of multiple words in the input texts (e.g. "5days"), and some token strings should be treated as an entirety (e.g. "Thanksgiving day"). To deal with this problem, we use some generic rules listed as follow.

- *Multi-tokens that should be combined as one.* In most situation, we concatenate their lemmas as pseudo lemma of the combined token, and use the type of it as a pseudo POS-tag.
  - *Holidays.* A set of common holidays mainly collected from other recognizing systems like SUTime, UWTime and SynTime. Including "New Year", "Valentine Day" and so on.
  - *Numeral values written as English word(s).* We normalize these strings and use the numeral value as the pseudo lemma. Some special values will be typed as their corresponding types directly (e.g. "nineteen ninety-six" will be typed as "YEAR"). Others will be tagged with the POS-tag "CD".
  - *Some common combination of words representing indefinite quantity.* Including phrases like "a few", "dozens of".
  - *Some common combination of words representing inequality among quantities.* Including phrases like "more than", "no less than", "at least".
- *Single token that should be split.* Words containing a "-" or "/" will be split at the position where they appear before tokenization. After annotating by CoreNLP (Manning et al. 2014), if a token is a concatenation of a number and an English character string, and cannot match any of our types, it will be split to 2 tokens. The number is tagged as "CD", the other parts will be re-lemmatized and inherits the original POS-tag. For example, "5mins/NNS" will become "5/CD" and "min/NNS", and "1990s/NNS" will not be split since it has already matched the DECADE type.

After pre-processing, the inputs are converted to token sequences. We generate candidate patterns from the training time expressions, then use an optimization method to select patterns. For the range of precision-control parameter ρ, we suggest to set it in the range of 0.8 to 0.98 in real applications, depends on your demands and the quality of your training dataset(s). Moreover, if the optimizing objective has been clarified (e.g. maximizing the F_1 score), the value of ρ can be determined by a validation procedure on a development set or cross-validation on your training set. Formal definition and detailed explanation about our implementation of pattern generation and selection will be presented in section 4 and 5.

3.2 The Test process

In the test process, PTime reads the test documents and does the same pre-processing as the training process to transform them to token sequences. With the patterns previously selected in the training process, PTime scans the transformed token sequences and extracts time expressions from them. After some necessary post-processing, PTime outputs the annotated documents.

Post-processing After extraction, adjacent and overlapped strings will be merged as a longer expression. Besides, as the expressions of ranges are annotated as two separated time expressions in the TIMEX3 standard (Pustejovsky et al. 2005) (e.g. "*⟨T⟩1957⟨/T⟩ and ⟨T⟩58⟨/T⟩*"), our approach may lose some numeral expressions which depend on nearby expressions. Several heuristic rules are designed to recognize them back. The post-processing is similar to other systems like TOMN.

4 Generating Candidate Patterns

In the training process, text are inputed as annotated documents, where each time expression is marked as an XML element. After pre-processing, we treat each document as a token string, and each token can be identified by the document it belongs and its position in the document. For example, the head token in the first time expression of 7th training document (shown in figure 3) can be represented as $tok_{\{7,1\}} = (\text{“}the\text{”}, \text{“}DT\text{”})$, where "the" and "DT" are the lemma and the POS-tag of the token respectively. Similarly, we identify a string by its document id, start position and end position. For example, the appearance of the time expression "the start of 2015" in that document can be denoted as $str_{\{7,1,4\}} = (tok_{\{7,1\}}, \ldots, tok_{\{7,4\}})$.

Celebrating ⟨T⟩the start of 2015⟨/T⟩ at the float!!
Is anyone here as well? Waiting for Sun Yanzi and ...

Figure 3: A fragment of the 7th document in Tweet-training

By transforming each token to its corresponding type, we can abstract each string to a sequential pattern.

Definition 1 (Token Type) *A token type is a tuple consisting a type name and a regex expression, i.e.*

$$type_i = (name_i, regex_i)$$

In real applications, we require our types to be non-overlapped,

$$type(tok) = type_i.name$$

iff

$$tok.lemma/tok.pos \text{ matches } type_i.regex$$

Some examples of the token types are shown in table 1.

Table 1: Some examples of our token types

Name	Content
DEFINITE_DET.	(the | this | that | these | those)/W?DT
EARLY_LATE	(earlier | later)/RBR? | (before | after | ago | early | late)/RB
TIME_UNIT	(second | minute | hour | ... | century | era)/NNP?S?

Definition 2 (Pattern) *The pattern of a string is defined as the sequence of token types generalized from each corresponding token of it. i.e.*

$$pat\left(str_{\{i,l,r\}}\right) = \Big(type\left(tok_{\{i,l\}}\right), type\left(tok_{\{1,l+1\}}\right), \ldots$$
$$\ldots, type\left(tok_{\{i,r\}}\right)\Big)$$

We denote the set of time expressions in the training texts as E, and abstract the strings in E to get a candidate pattern set P, i.e.

$$P = \{pat(e) | e \in E\}$$

There are two main problems need to be considered in generating candidates patterns. One is the balance between generality and specificity of types. A good type design should have the generality to ensure the generated patterns can recognize newly appeared time expressions with an acceptable accuracy. Another one is the difficulty of accommodating all potential time-related tokens with manually designed types. A previous study (Zhong, Sun, and Cambria 2017) has shown that domain/dataset based type expansion does improve the performance of recognition.

4.1 Token Types

Designing a type system is a very difficult task which need tremendous domain knowledges and experience. Prior studies have designed many good type systems and verified their types on benchmark datasets. Our type system contains 32 fine-grained types classified from the perspective of POS-tags and semantic functions. Most of the types and their corresponding regexs are collected from SUTime[1] and SynTime[2]. We subdivided some types in a finer granularity and removed some informal or ambiguous words (e.g. remove the abbreviation "hr" for "hour" because of it is also the abbreviation for "human resources"). The aim of fine-grained classification and adjustment of types is to reduce the amount of probable mistakes caused by over-generalization. A detailed description of the types is attached in the appendix.

4.2 Untyped Tokens

Because of the probable incompleteness of the token types, some tokens may not match any types, like "fiscal" in the TempEval-3 dataset and "war" in the WikiWars dataset. Statistics on training dataset show that there are only 40 to 60 words cannot be typed by our types in each dataset. Therefore, we let the untyped tokens remain unchanged. In other words, we dynamically create one-token types for the untyped tokens. For example, the pattern of the time expression "1 hr" is "NUMBER *hr*", where "*hr*" is a dynamically created one-token type with a regex "hr/[\^s]+", which can only match the word itself. This technique can give our approach the adaptability to various data without over-generalization.

5 Selecting Patterns

After generation, we get the candidate pattern set P, which can extract string from the token sequences of input documents.

Definition 3 (Strings Matched by a Pattern) *For a pattern $p \in P$, on given input documents D, we define $S(p)$ as the set of strings that can be matched by types in p in sequence.*

$$S_D(p) = \left\{ str_{\{i,l,l+|p|-1\}} \,\middle|\, \bigwedge_{k=0}^{|p|-1} type\left(tok_{\{i,l+k\}}\right) = p_k \right\}$$

However, as previously discussed in section 1, some patterns are over generalized so they will mistakenly extract non-timex strings. In extreme cases, we might have $|\bigcup_{p\in P} S_{training}(p)| \gg |E|$. Therefore, we wish to select an high quality subset $Q \subseteq P$ instead of using the whole P, which can still recognize most of correct time expressions.

To ensure the quality of the selected pattern set Q, we measure the quality of each candidate pattern on training texts by a cost function, which is computed by counting the

[1]https://github.com/stanfordnlp/CoreNLP/tree/master/src/edu/stanford/nlp/time/rules

[2]https://github.com/xszhong/syntime/tree/master/syntime/resources/syntimeregex

number of mistakes caused by it (i.e. the number of strings which are extracted by the pattern but not "contained" in the time expression set E).

Definition 4 (Containing Relationship) *We say a string $str_{\{i,l,r\}}$ is "contained" in a time expression set X, if and only if it is contained by an expression in X. i.e.*

$$str_{\{i,l,r\}} \Subset X$$

iff

$$\exists l_x, r_x \left(l_x \le l \land r \le r_x \land str_{\{i,l_x,r_x\}} \in X\right)$$

Definition 5 (Cost Function) *For any $p \in P$, its cost is defined on the training data as*

$$Cost(p) = \sum_{s \in S_{training}(p)} \begin{cases} 0 & s \Subset E \\ 1 & \text{otherwise} \end{cases}$$

The total cost of a set $Q \subseteq P$ is simply defined as summing up the cost of each pattern in it, i.e.

$$Cost(Q) = \sum_{p \in Q} Cost(p)$$

We assume the limitation of mistakes should be related to the number correctly recognized time expressions $|E|$. To fit different precision-recall requirements, we use a parameter ρ to define an adjustable bound.

Definition 6 (The Cost Bound with the parameter ρ) *The total cost should not exceed a bound B, which is defined as*

$$B = |E| \cdot (1 - \rho)$$

where $\rho \in [0, 1]$.

A higher ρ may improve the precision of selected patterns by limiting the number of mistakes caused by them, but may lose some useful patterns.

To measure the gain of selected patterns, a straightforward way is to count the number of time expressions a pattern can match. However, the strict match requirement will underrates short patterns, as it ignores the fact that a long expression might be a composition of several short expressions (e.g. "10 a.m. Saturday" is composed by "10 a.m." and "Saturday"). In order to assess the patterns more accurately, we measure the contribution of selected patterns by defining a coverage function for patterns on each time expressions.

Definition 7 (Coverage Function) *For a pattern $p \in P$ and a tokened expression $e \in E$, we say p has a positive coverage $Cov(p, e)$ on e if and only if p can match a substring of e (or the whole e).*

$$Cov(p, e) = \begin{cases} \frac{|p|}{|e|} & \exists s \in S_{training}(p)\ s \text{ is a substring of } e \\ 0 & \text{otherwise} \end{cases}$$

For example, Cov("ORDINAL WEEK", "the/DT second/JJ Sunday/NNP") ≈ 0.67 and Cov("ORDINAL WEEK", "Friday/NNP") $= 0.0$.

With the coverage function, we could define the gain of a set of selected patterns as follows.

Definition 8 (Gain Function) *For a set $Q \subseteq P$ of patterns, its gain over the time expression set E is defined as*

$$Gain(Q) = \sum_{e \in E} \max_{p \in Q} \{Cov(p, e)\}$$

Finally, the problem of selecting patterns can be formalized as

Problem 1 (Pattern Selection) *Given the candidate pattern set P, training documents D, time expression set E and a control parameter ρ, Select a subset $Q \subseteq P$ to*

$$\begin{array}{ll} \text{Maximize} & Gain(Q) \\ \text{Subject to} & Cost(Q) \le B \end{array}$$

This optimization problem is a case of maximizing a set coverage function with a linear constraint, which has some well-known instances like the Set Cover problem, the Maximum Coverage problem and the Budgeted Maximum Coverage problem (Khuller et al. 1999). And our problem definition is an alternation of the Extended Budgeted Maximum Coverage problem (Ding et al. 2015). Efficient approximation algorithms of it has been sufficiently discussed in (Wolsey 1982). Follow the prior studies, we solve our pattern selection problem with algorithm 1.

Algorithm 1: Algorithm for Pattern Selection

```
Input: the candidate pattern set $P$, time expression set
       $E$ and the functions $Cov$, $Cost$
Output: A subset $Q \subseteq P$ denotes the selected patterns
$Q_1 \leftarrow$ GreedySelect($\emptyset$);
$p^* = \text{argmax}_{p \in P}\{Gain(\{p\})\}$;
if $Cost(p^*) > |E| \cdot (1 - \rho)$ then
    return $Q_1$;
$Q_2 \leftarrow$ GreedySelect($\{p^*\}$);
if $Gain(Q_1) > Gain(Q_2)$ then
    return $Q_1$;
else
    return $Q_2$;
```

The sub-procedure *GreedySelect* is described as algorithm 2. In each iteration, the algorithm re-ranking the remain candidates pattern set R, find the pattern p_i with a maximum gain-cost ratio and try to add p_i to the selected pattern set. The micro value $\varepsilon \ll 1$ in $Cost(p_i) + \varepsilon$ is just a technique trick to avoid the probable divide-zero error, which will not affect the choice of the algorithm.

Our algorithm 1 is slightly different with the classic version. We choose a better solution between *GreedySelect*($\emptyset$) and *GreedySelect*($\{p^*\}$), but the classic algorithm chooses a better solution between *GreedySelect*($\emptyset$) and $\{p^*\}$. This difference affects neither the approximation ratio nor the time complexity, since $Gain(GreedySelect(\{p^*\}) \ge Gain(\{p^*\})$ always holds and the theoretical complexity of *GreedySelect*($\{p^*\}$) is the same as *GreedySelect*($\emptyset$). According to prior proofs, our algorithm has the time complexity $O(|P|^2|E|)$ (Ding et al. 2015) and the approximation ratio of $1 - e^{-\beta} \approx 0.35$ (Wolsey 1982), where β is the unique root of $e^x = 2 - x$.

Algorithm 2: GreedySelect

```
Input: Initial selected patterns $I$, and all of the input of
       algorithm 1
Output: A subset $Q \subseteq P$ denotes the selected patterns
$R \leftarrow P - I$;
$Q \leftarrow I$;
repeat
    $p_i \in R \leftarrow \text{argmax}\{\frac{(Gain(Q \cup \{p_i\}) - Gain(Q))}{Cost(p_i) + \varepsilon}\}$;
    if $Cost(Q \cup \{p_i\}) \leq |E| \cdot (1 - \rho)$ then
        $Q \leftarrow Q \cup \{p_i\}$;
    $R \leftarrow R \setminus \{p_i\}$;
until $R = \emptyset$;
return $Q$
```

6 Evaluation

We evaluate PTime against six state-of-the-art approaches (i.e., HeidelTime, SUTime, SynTime, ClearTK, UWTime, and TOMN) on three datasets namely TempEval-3, WikiWars, and Tweets.

6.1 Experimental Setup

Datasets We evaluate our approach on the TempEval-3 (UzZaman et al. 2013), the WikiWars (Mazur and Dale 2010) and the Tweets (Zhong, Sun, and Cambria 2017). For the TempEval-3, we use the training and test sets splits following previous studies (Bethard 2013) i.e. use the TimeBank (Pustejovsky et al. 2003) corpus as the training set and the platinum-annotated corpus as the test set. For the WikiWars and the Tweets, we follow previously work to set the split of training and test sets (Lee et al. 2014; Zhong, Sun, and Cambria 2017). For development, we perform a 10-fold cross-validation on each training dataset. Detailed statistics of the datasets is presented in table 2[3].

Table 2: Dataset statistics

Dataset	#Docs	#Words	#Timex
TimeBank	183	61,418	1,243
TempEval-3 (test)	20	6,375	138
WikiWars (whole)	22	119,468	2,671
Tweets (whole)	942	18,199	1,127

Comparison Methods We compare our system primarily to six state-of-the-art approaches, including HeidelTime (Strötgen and Gertz 2010; Strötgen et al. 2014), SUTime (Chang and Manning 2012), SynTime (Zhong, Sun, and Cambria 2017), ClearTK (Bethard 2013), UWTime (Lee et al. 2014) and TOMN (Zhong and Cambria 2018). HedialTime, SUTime and SynTime are rule based approaches. UWTime is a semantic parsing approach based on a combinatory categorial grammar. ClearTK and TOMN are black-boxed approaches with CRFs frameworks. The results of all the comparison approaches are cited from (Zhong and Cambria 2018).

[3]The statistics of TimeBank, WikiWars, Tweets are cited from (Zhong and Cambria 2018), and the statistics of the test set of TempEval-3 is cited from (Lee et al. 2014)

Evaluation Metrics We use the evaluation toolkit of TempEval-3 (UzZaman et al. 2013) to measure the results of our approach. We report the $Precision$, the $Recall$, and the F_1 score in terms of *strict match* and *relaxed match*. The strict match means exact match between the extracted time expressions and the ground truth while the relaxed match means that there exists overlap token(s) between them.

Parameter Settings We grid searched the value of ρ with a step of 0.01 for maximizing the strict match F_1 score on each dataset. The values of ρ are set to 0.87, 0.94, 0.94 for TempEval-3, WikiWars and Tweets respectively.

6.2 Experimental Results and Analysis

Experimental Results Table 3, table 4 and table 5 report the performance of PTime on given ρ and comparison approaches. The best results are in **bold face** and the second are underlined.

Table 3: Test results on TempEval-3

Method	Strict Match			Relaxed Match		
	$Pr.$	$Re.$	F_1	$Pr.$	$Re.$	F_1
HeidelTime	83.85	78.99	81.34	93.08	87.68	90.30
SUTime	78.72	80.43	79.57	89.36	91.30	90.32
ClearTK	85.90	79.70	82.70	93.75	86.96	90.23
UWTime	86.10	80.40	83.10	94.60	88.40	91.40
SynTime	91.43	**92.75**	**92.09**	94.29	**95.65**	**94.96**
TOMN	**92.59**	90.58	91.58	**95.56**	93.48	94.51
PTime	85.19	83.33	84.25	92.59	90.58	91.58

Table 4: Test results on WikiWars

Method	Strict Match			Relaxed Match		
	$Pr.$	$Re.$	F_1	$Pr.$	$Re.$	F_1
HeidelTime	**88.20**	78.50	83.10	95.80	85.40	90.30
SUTime	78.61	76.69	76.64	95.74	89.57	92.55
ClearTK	87.69	80.28	83.82	96.80	90.54	93.56
UWTime	87.70	78.80	83.00	**97.60**	87.60	92.30
SynTime	80.00	80.22	80.11	92.16	92.41	92.29
TOMN	84.57	80.48	82.47	96.23	92.35	94.25
PTime	86.86	**87.57**	**87.21**	95.98	**96.76**	**96.37**

Table 5: Test results on Tweets

Method	Strict Match			Relaxed Match		
	$Pr.$	$Re.$	F_1	$Pr.$	$Re.$	F_1
HeidelTime	91.67	74.26	82.05	96.88	78.48	86.71
SUTime	77.69	79.32	78.50	88.84	90.72	89.77
ClearTK	86.83	75.11	80.54	96.59	83.54	89.59
UWTime	88.36	70.76	78.59	97.88	78.39	87.06
SynTime	89.52	94.07	91.74	93.55	98.31	95.87
TOMN	90.69	**94.51**	92.56	93.52	97.47	95.45
PTime	**92.92**	94.09	**93.50**	**97.92**	**99.16**	**98.53**

The results show that PTime outperforms all other approaches on WikiWars and Tweets, and takes a third place in both strict match F_1 and relaxed match F_1 on TempEval-3. On TempEval-3, PTime is inferior to SynTime and its succeeding work, TOMN, both of which significantly outperformed other systems at least 7% in strict match F_1 score. On WikiWars, our approach outperforms all other approaches in recall and F_1 without obviously loss in precision. It improves the strict match F_1 score by 3.39% and the relaxed match F_1 score by 2.12% comparing to the second best one. On Tweets, we raise the strict match F_1 by 0.94% and outperforms all other systems in all the three of the relaxed match precision, recall and F_1 score, including a 2.66% improvement in the relaxed match F_1 score. Besides, our results maintain a better balance between precision and recall, as the differences between them are relatively small in our results comparing with others.

Besides, The running efficiency of PTime is in the same order comparing to other approaches according to (Ning et al. 2018). The generation of candidate patterns can be finished within 1 minute, running the selection step took 7 minutes in average, and recognizing timexes from test set finished within 3 seconds in average. The above results are achieved by a simple implementation of PTime written by Java and Scala without any optimization, running on a personal workstation with an Intel E3-1226 CPU of 3.30GHz.

Result analysis To analyze our results in details, we compared our results with the best performed rule-based approach SynTime. The results of SynTime are generated by using the source code published by its authors [4].

We achieved good results on WikiWars and Tweets. One reason is our approach can generate patterns with rich structures, such as "NUMBER TIME_UNIT EARLY_LATE", so we can recognize the exact time expression from the sentence *"Russia declared war on Japan ⟨T⟩eight days later⟨/T⟩."*, while SynTime missed the suffix "*later*" because its heuristic rules do not take it as a suffix modifier. PTime can recoginize the time expression "*late in 1985*" with the pattern "EARLY_LATE RELATION YEAR", while not mistaken the border of "*1,460 years*" in the sentence "*The first Roman Emperor in ⟨T⟩1,460 years⟨/T⟩ raised*". The patterns of PTime can handle some ambiguity of words. For example, though "hr" is not included in any of its types, PTime selected a reasonable and useful pattern "NUMBER *hr*" for the Tweets dataset, while the deterministic type based approaches might mistakenly recognize the "hr" from the phrase "*Singapore's HR Manager*".

However, the performance of PTime has a large gap to SynTime on TempEval-3, especially in recall. A main reason is that PTime failed to recognize some one-token time expressions like "*years*", "*days*", i.e. PTime did not select the pattern "TIME_UNIT". We observed the training data and found out some negative samples like "*37 years old*" or "*three - quarters*". These negatives make PTime overrated the cost of the pattern "TIME_UNIT", which leads to the result that it was dropped in selection. This is because PTime can not utilize the contextual information outside a string for disambiguation. Besides, as a data-driven approach, the incompleteness of training set also affected the recognizing ability of PTime. For example, PTime failed to recognize the expression "3:07:35", because PTime did not see its corresponding pattern "TIME" in the training data. If we manually add these common patterns to the selected pattern set, the performance is dramatically improved.

Besides, analysis of the selected patterns showed that there are many meaningful common patterns, including patterns like "TIME TIME_REFERENCE" and "ORDINAL MONTH YEAR". It is no doubt that adding the common patterns to PTime can further improve its performance. For example, the common part of selected patterns from the WikiWars and Tweets training datasets have 43 patterns, 14 of which are not appeared in the patterns selected from the TimeBank. If we simply treat these patterns as "common patterns well known to human experts", the F_1-score of PTime on TempEval-3 can be promoted to 87.54%. PTime benefits the formation of common patterns, which in turn can enhance the performance of PTime [5].

7 Conclusion

The contributions of this paper are summarized as follows:

- We proposed a pattern-based approach to recognizing time expressions, called PTime. In this approach, sequential patterns are generated from time expressions in training text by using their token types, and the pattern selection is optimized by using the Extended Budgeted Maximum Coverage (EBMC) model. Especially, a parameter is introduced to meet various precision-recall demands in different applications.
- We implemented the PTime approach, especially, we manually designed 32 fine-grained token types, and designed a greedy algorithm for selecting patterns based on EBMC model.
- We evaluated our implementation on three benchmark datasets. The experimental results showed that our approach achieved best F-Measure on two datasets, as compared with state-of-the art approaches. It improved the strict match F_1 score by 3.39% on the WikiWars dataset and 0.94% on the Tweets dataset comparing with the second best approach.

In the near future, we will explore (semi-)automatic ways of designing a better token type system. We are also interested in enhancing PTime by including common patterns well known to human experts. Furthermore, it is also interesting to combine PTime with classic time expression normalization techniques for better understanding temporal informations.

8 Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 61772264). We thank our team member Hao Zhang for inspiring discussions at the early stage of the research.

[4]https://github.com/xszhong/syntime

[5]The detailed results including lists of selected patterns can be found at http://ws.nju.edu.cn/ptime

References

Angeli, G., and Uszkoreit, J. 2013. Language-independent discriminative parsing of temporal expressions. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 83–92.

Bethard, S. 2013. Cleartk-timeml: A minimalist approach to tempeval 2013. In *Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics*, volume 2, 10–14.

Chang, A. X., and Manning, C. D. 2012. Sutime: A library for recognizing and normalizing time expressions. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, volume 2012, 3735–3740.

Ding, J.; Ding, W.; Hu, W.; and Qu, Y. 2015. An ebmc-based approach to selecting types for entity filtering. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 88–94.

Hobbs, J. R., and Pan, F. 2006. Time ontology in owl. *W3C working draft* 27:133.

Hobbs, J. R.; Appelt, D.; Bear, J.; Israel, D.; Kameyama, M.; Stickel, M.; and Tyson, M. 1997. Fastus: A cascaded finite-state transducer for extracting information from natural-language text. *Finite-state language processing* 383–406.

Khuller, S.; Moss, A.; Naor, J.; et al. 1999. The budgeted maximum coverage problem. *Information Processing Letters* 70(1):39–45.

Lee, K.; Artzi, Y.; Dodge, J.; and Zettlemoyer, L. 2014. Context-dependent semantic parsing for time expressions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, 1437–1447.

Mani, I., and Wilson, G. 2000. Robust temporal processing of news. In *Proceedings of the 38th annual meeting on Association for Computational Linguistics*, 69–76. ACL.

Manning, C.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S.; and McClosky, D. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 55–60.

Mazur, P., and Dale, R. 2010. Wikiwars: Wikiwars: A new corpus for research on temporal expressions. In *Proceedings of the 15th Conference on Empirical Methods in Natural Language Processing*, 913–922. ACL.

Ning, Q.; Zhou, B.; Feng, Z.; Peng, H.; and Roth, D. 2018. Cogcomptime: A tool for understanding time in natural language. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 72–77.

Pustejovsky, J.; Hanks, P.; Sauri, R.; See, A.; Gaizauskas, R.; Setzer, A.; Radev, D.; Sundheim, B.; Day, D.; Ferro, L.; et al. 2003. The timebank corpus. In *Corpus linguistics*, volume 2003, 40. Lancaster, UK.

Pustejovsky, J.; Knippen, R.; Littman, J.; and Saurí, R. 2005. Temporal and event information in natural language text. *Language resources and evaluation* 39(2-3):123–164.

Pustejovsky, J.; Lee, K.; Bunt, H.; and Romary, L. 2010. Iso-timeml: An international standard for semantic annotation. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, volume 10, 394–397.

Strötgen, J., and Gertz, M. 2010. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, 321–324. ACL.

Strötgen, J.; Bögel, T.; Zell, J.; Armiti, A.; Van Canh, T.; and Gertz, M. 2014. Extending heideltime for temporal expressions referring to historic dates. In *Proceedings of the 9th International Conference on Language Resources and Evaluation*, 2390–2397. Citeseer.

Strötgen, J.; Zell, J.; and Gertz, M. 2013. Heideltime: Tuning english and developing spanish resources for tempeval-3. In *Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics*, volume 2, 15–19.

UzZaman, N.; Llorens, H.; Derczynski, L.; Allen, J.; Verhagen, M.; and Pustejovsky, J. 2013. Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *Proceedings of the 2nd Joint Conference on Lexical and Computational Semantics*, volume 2, 1–9.

Verhagen, M.; Mani, I.; Sauri, R.; Knippen, R.; Jang, S. B.; Littman, J.; Rumshisky, A.; Phillips, J.; and Pustejovsky, J. 2005. Automating temporal annotation with tarsqi. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, 81–84. ACL.

Verhagen, M.; Sauri, R.; Caselli, T.; and Pustejovsky, J. 2010. Semeval-2010 task 13: Tempeval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, 57–62. ACL.

Wolsey, L. A. 1982. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research* 7(3):410–425.

Zhong, X., and Cambria, E. 2018. Time expression recognition using a constituent-based tagging scheme. In *Proceedings of the 27th World Wide Web Conference on World Wide Web*, 983–992. WWW.

Zhong, X.; Sun, A.; and Cambria, E. 2017. Time expression analysis and recognition using syntactic token types and general heuristic rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, volume 1, 420–429. ACL.