A Novel Probabilistic Model for Dependency Parsing

Shixi Fan, Xuan Wang, Xiaolong Wang

School of Computer Science
Harbin Institute of Technology Shenzhen Graduate School
C303c, Xili university town, ShenZhen, China
fanshixi@hit.edu.cn, { wangxuan, wangxl}@insun.hit.edu.cn

***Abstract*—A new knowledge based probabilistic dependency parsing (KPDP) is presented to overcome the local optimization problem of native probabilistic models. KPDP is composed of two stages: (1) selecting a set of constituent parse trees with an extensive bottom-up chart parsing algorithm which employs Maximum Entropy Models to calculate single arc probabilities; (2) finding the best parsing tree with the help of word knowledge. Different from previous studies, we incorporate word knowledge into parsing procedure. Based on case grammar theory, the word knowledge is represented as some patterns which group those arcs with the same head. Thus, the KPDP contains both single arc information and the relationship information between relevant arcs. KPDP is evaluated experimentally using the dataset distributed in CoNLL 2008 share-task. An unlabelled arc score of 87 % is reported, which is 3.39% higher than the native model without word knowledge. This work will contribute to and stimulate other researches in the field of parsing.**

***Index Terms*—knowledge based probabilistic dependency parsing, Chart Parsing, Maximum Entropy Models**

I. Introduction

Syntactic dependency parsing was firstly introduced by Lucien Tesnière in 1959 [1] and gained wide interests recently [2-4] for its wieldy application in machine translation, shallow semantic analyzing, question answering system and etc. A dependency parse tree represents words and their modifiers through labeled directed arcs under constraints that each word can modify only one other word, as shown in Fig. 1. An arc is a word pair $(w_i - > w_j)$ representing that w_j is the modifier or dependant of w_i.

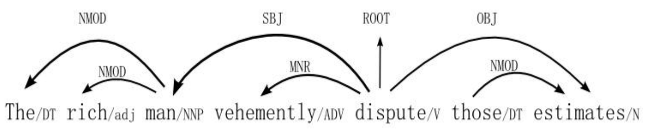

NMOD
NMOD
SBJ
MNR
ROOT
OBJ
NMOD
The/DT rich/adj man/NNP vehemently/ADV dispute/V those/DT estimates/N

Figure 1. Example of a dependency parse tree.

Arcs with specific labels in the parse tree are used to indicate the relations between words. For some words such as adjectives and articles, one single arc is sufficient to represent its meaning in a sentence, such as the arc (man ->rich) with label NMOD in Fig. 1. But for nouns and verbs, their meaning may not be explained by a single arc. For example, the meaning of *dispute* should be represented by two arcs which indicate its subject and object. Therefore, both arcs and their relationships should be considered in the parsing procedure. Unfortunately, most parse models such as CKY [5], Deterministic algorithm [6], Maximum Spanning Trees [7] ignore the relationship between arcs and treat each arc solely. Thus, those models all suffer the local optimization problem.

Currently, conditional parsing models appear to have culminated in large margin training approaches [8], which get state-of-the-art performances in English dependency parsing. The large margin approach could give global optimized results. However, it is difficult to focus on errors of any particular arc, and linguistic knowledge cannot be integrated easily.

This paper presents a new approach for dependency parsing which considers both arcs and their relationships. Unlike native probabilistic and deterministic models that ignore the relationship of the arcs, our approach uses the principle of case grammar and integrates word knowledge into the parsing procedure.

The rest of the paper is structured as follows: Section 2 describes the KPDP parsing algorithm; section 3 explains the probability model for arcs. Section 4 is about experiments and discussion. Conclusions are stated in section 5.

II. KPDP Parsing Algorithms

A. Finding the constituent parser tree set

The dependency parser tree T for a sentence with n words $W = (w_1, w_2, ..., w_n)$ can be represented as a set of directed arcs:

$$T = \{(w_i - > w_j); w_i \in W \ \& \ w_j \in W\} \tag{1}$$

Where $(w_i -> w_j) \in W \times W$ is a directed arc of T, and w_j depends on w_i. Given a dependency label set R, each arc is reflected into a dependency type as $f: T -> R$. For native probabilistic dependency parser, arcs are independent of each other and their probabilities can be calculated as:

$$p(w_i -> w_j) = \theta^{\mathrm{T}} f(w_i, w_j) \quad (2)$$

Where $f(w_i, w_j)$ is a feature vector selected over w_i and w_j, and θ is the corresponding weight vector. How to generate features and how to determine their weights will be described in section 3. Let $\Phi(T)$ denotes all possible dependency parse trees:

$$|\Phi(T)| = |W|^{|W|} \quad (3)$$

Eq. (3) indicates that the size of $\Phi(T)$ is exponential of sentence length. It is intractable to find the best parse tree using enumeration algorithm because of the expensive computation. To overcome this problem, we design an extensive chart parse algorithm to filter $\Phi(T)$ into a computable set. The extensive bottom-up chart parse algorithm is stated as follows:

Definition 1: Chart [i, j] stores all the constituent part trees from words i to j
Definition 2: threshold is a variable in range (0, 1) which is used to filter those arcs with probability lower than the threshold.

```
Extensive chart parse (W, threshold) {
    n=|W|
    set all Chart[i， j] to null
    For i in (1, n) {   // Initialization
        If  P (W[i] -> W[i+1]) > threshold
        Add (W[i] -> W[i+1]) into Chart [i, i+1]
        If  P (W[i+1] -> W[i]) > threshold
        Add (W[i+1] -> W[i]) into Chart [i, i+1]
    }
    For i in (2, n-1):
    For j in (1, n-i+1):
    For k in (1, i):
    For T1 in Chart [j, j+k]:
    For T2 in Chart [j+k, j+i] {
            Ret =Merge (T1, T2, threshold)
            Add Ret into Chart [j, j+i]
            }
    Return Chart [1, n]
}
```

The "Extensive chart parse" function is an extensive chart parsing algorithm using down to up strategy. The function includes two main steps: (1) Initializing all the neighbor two words into part parser trees. (2) Forming all the part parsing trees step by step with length from 3 to the whole sentence. The parameter "threshold" is used to filter probable arcs in part parsing trees and only those arcs with probability higher than threshold are selected.

The child function "Merge (T1, T2, threshold)" merges two part parsing tree into a set of new part parsing trees which cover all the words in T1 and T2. Each part tree has a root word which does not depend on any other one. There are two kind merging trees: (1) the root of T1 depends on a word of T2; (2) the root of T2 depends on a word of T1. The maximum merging tree set size is |T1| +|T2|. The parameter threshold is used to filter a probable arc. The Merge function is stated as follows:

```
Merge (T1, T2, threshold) {
    Ret =null
    r1 = Root (T1) //the root of tree T1
    For w2 in Words (T2) {
     If  P(w2->r1)> threshold {
    ConstituentT= T1+T2 + $(w2 -> r1)$
    Add ConstituentT into Ret
    }}
    r2 = Root (T2) //the root of  tree T2
    For w1 in Words (T1) {
    If  P(w1->r2)> threshold {
    ConstituentT= T1+T2 + $(w1 -> r2)$
    Add ConstituentT into Ret
    }}
    Return Ret
}
```

According to the algorithm, the constituent parser tree set C is formed:

$$C = \{T \in \Phi(W) \mid \forall (w_i -> w_j) \in T, p(w_i -> w_j) > \delta\} \quad (4)$$

And the probability of a parser tree is:

$$P(T) = \prod_{(w_i -> w_j) \in T} p(w_i -> w_j) \quad (5)$$

E.q. 5 calculates arc probability independently without considering the relationships between arcs.

B. Word knowledge for dependency parsing tree

Case Grammar is a system of linguistic analysis, focusing on the link between the valence of a verb and the grammatical context it requires, created by the American linguist Charles J. Fillmore in (1968) [9]. This theory analyzes the surface syntactic structure of sentences by studying the combination of deep cases (i.e. semantic roles) -- Agent, Object, Benefactor, Location or Instrument -- which are required by a specific verb. For instance, the verb "give" in English requires an Agent (A) and Object (O), and a Beneficiary (B); e.g. "Jones (A) gave money (O) to the school (B). Take sentence "**I give the bag to jack**" as an example: The Agent is "I", Object is "the bag" and the Beneficiary is "to jack". The dependency parsing tree for the sentence is shown in Fig. 2.

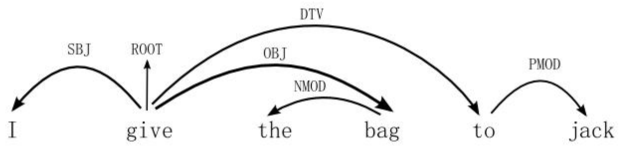


Figure 2. Dependency parsing tree for verb 'give'.

Fig.2 shows that verb ***give*** monitors **I**, **bag** and **to**. On the other hand, these three words all depend on verb ***give***. That is, the case grammar is consistent with dependency parser. According to the theory of case grammar, words can be represented as some case structures. And more importantly, the case structure can describe the relationship of those arcs which have the same center word in a dependency parsing tree.

Our work is similar to (Kun Yu et al) who used case structures to construct Chinese dependency parser and got good result [10]. Our contributions are a general word pattern representation for word knowledge and a POS pattern. Case structure is a kind of word knowledge for dependency parsing. However, case structure can not be used directly for two reasons: (1) Case structure is too restrictive for a dependency parsing. (2) Case structure only considers verb. In this study, we use word patterns to represent word knowledge. Future more, we relax verb into all words. A word pattern for a specific word w is a tuple $K =< [p_n, p_{n-1},....p_1]_L, w,[p_1, p_2,....p_m]_R >$ which indicates that there are *n* words on the left of w and *m* words on the right. p_i is the POS tag of dependants. Take sentence in Fig.1 as an example, word pattern for 'dispute' is <[NNP,ADV],dispute,[N]> and the word pattern for 'man' is <[DT,ADJ],man,[]>. Obviously, a word may have more than one case structure, and then there will be more than one pattern accordingly. The pattern probability can be calculated statistically:

$$P(k_i \mid w) = \frac{count(k_i)}{\sum_{w \in k_j} count(k_j)} \tag{6}$$

According to Zipf's law, the frequency of any word is inversely proportional to its rank in the frequency table. Thus, the most frequent word will occur approximately twice as often as the second most frequent word, which occurs twice as often as the fourth most frequent word. Therefore, the case structure pattern probability will encounter the data sparseness problem. To overcome this problem, we generate the word pattern to POS pattern:

$$KP =< [p_k, p_{k-1},....p_1]_L, p,[p_1, p_2,....p_m]_R > \tag{7}$$

The POS pattern use POS tag to replace the word in the word pattern. Then POS pattern can be transformed from word pattern. For example, The word pattern <[NNP,ADV],dispute,[N]> from Fig.1 can be transformed into <[NNP,ADV],v,[N]>. By this way, POS patterns group all these word patterns with the same POS tags and their probability can also be calculated statistically:

$$KP(k_i \mid p) = \frac{count(kp_i)}{\sum_{p \in kp_j} count(kp_j)} \tag{8}$$

With the POS pattern, word pattern probability can be smoothed as:

$$P(k_i \mid w) = \theta * P(k_i \mid w) + (1-\theta) * KP(k_i \mid p), \quad 0 < \theta < 1 \tag{9}$$

Where p is the POS tag of word w and θ is a parameter which decides the impact of POS pattern on the case structure pattern.

C. Selecting best parser tree with word knowledge

When word knowledge is considered, the probability for a constituent tree Eq. (5) is modified as:

$$P(T) = \prod_{(w_i -> w_j) \in T} p(w_i -> w_j) * \prod_{k_i, w \in T} p(k_i \mid w) \tag{10}$$

E.q. 10 contains two kind of probability: single arc probability and pattern probability which calculated statistically. Finally, **KPDP** gives the best parse tree:

$$T^* = \arg\max_{T \in C} \quad \mathrm{P}(T) \tag{11}$$

III. Arc Probabilistic Model for KPDP

A. Maximum entropy model

The major problem of the extensive bottom-up chart parsing is how to get the arc probability $P(w_i -> w_j)$. The ME (Maximum Entropy) framework, presented by Berger et al. [11], has two benefits for a parser builder [12]. First, factoring the probability computation into a sequence of values corresponding to various 'features" which suggests that the probability model can be easily changeable – just change the set of features used. Second the features used in ME models have no particular independence of each other.

The maximum entropy framework aims to "model all that is known and assume nothing about that which is unknown." This is achieved by choosing the model that fits all the constraints expressed by the training data. The features f is binary functions of two arguments. The expected value of *f* with respect to the empirical distribution is exactly the statistics we are interested in. The expected value is:

$$\tilde{P}(f) = \sum_{x,y} \tilde{P}(x, y) f(x, y)) \tag{12}$$

The expected value of *f* with respect to the model is:

$$P(f) = \sum_{x,y} \tilde{P}(x) P(y \mid x) f(x, y) \tag{13}$$

Given a set of feature functions, and a set of labeled training instances, we can formulate the following constraint equation:

$$P(f) = \tilde{P}(f) \tag{14}$$

I.e. the expected values of the model and of the training sample must be the same. The mathematical measure of the uniformity of a conditional distribution P(Y|X) is provided by the conditional entropy :

$$H(Y \mid X) = P(X,Y) * \sum P(Y \mid X) \tag{15}$$

Here marked as H(P). The Maximum Entropy Principle:

$$P_* = \arg\max_{P \in C} H(P)$$
$$C = \{P \in \mathcal{P} \mid P(f_i) = \tilde{P}(f_i), i = 1,2,...,n\} \tag{16}$$

To solve this constraining problem, we get the unique solution:

$$P(x) = \frac{1}{Z} \exp\left(\sum_i \lambda_i f_i(x)\right) \tag{17}$$

Where λ_i are the feature weights that need to be estimated from the training data and Z is a normalization factor to ensure P is a probability distribution.

B. Arc probabilistic model for KPDP

In our probabilistic model, x is a probable arc $(w_i -> w_j)$. $y \in \{R + 'null'\}$ where R is the dependency label set and 'null' indicates that x is not a valid arc. In this way, the ME model can predict the dependency arc and arc label simultaneously. For each arc $(w_i -> w_j)$, we select bag of words, bag of POS tags and position information as features. Table 1 lists part of feature templates.

TABLE I.
PART OF FEATURE TEMPLATES

W(i)	W(j)
P(i)	P(j)
P(i+1)	P(j+1)
P(i+2)	P(j+2)
P(i-1)	P(j-1)
P(i-2)	P(j-2)
Dis =(i-j)	
W(i) +W(j)	P(i)+ P(j)
W(i) +W(j)+Dis	P(i)+ P(j)+Dis
P(i)+ P(j)+ P(i-1)	P(i)+ P(j)+ P(i+1)
P(i)+ P(j)+ P(j-1)	P(i)+ P(j)+ P(j+1)
P(i)+ P(j)+W(i)	P(i)+ P(j)+W(j)
….	….

A dependency parser is a tree structure which can not easily be represented. Since a dependency arc only cover two words and a word can only depend on one other word or a dummy root. We can show a dependency parser tree structure in table form. Take sentence “terms are not disclose.” as an example and the data is shown in table 2. The first row is word sequence number starting from 1. The second and third row is word and POS tag sequence, respectively. The forth row is the arc head which is indicated by word sequence number defined in the first row. The last row is the arc label for the arc. By this way, a dependency parser structure and the sentence information are represented with a five row table.

TABLE II.
DEPENDENCY PARSER IN TABLE FORMAT

Word No.	1	2	3	4	5
word	terms	are	not	disclose	.
POS tag	NNS	VBD	RB	VBN	.
Arc head	2	0	2	2	2
Arc label	SBJ	ROOT	ADV	VC	P

To train a ME model, a series of (x, y) pairs and their features are needed as training instance. We denote a constituent arc as x, arc label as y and features are got from feature templates listed in table 1. Since a word may depend on any other word in the sentence or a dummy root, any probable instance should be considered in training and test procedure. For example, when word “terms” is considered, the instances for it are listed in table 3. The first column of table 3 is x, and the second column is y. The third column indicates the dependent word (or tail word) represented by the word sequence number and the fourth column indicates the head word of the arc. The first instance in table 3 is (Are->terms, SBJ) which is a real arc in the dependency tree. Arcs of the other three instances (Not->terms, disclose->terms and . ->terms) are not exit, so that their y are set to ‘null’. Features for these instances are not listed in the table.

TABLE III.
INSTANCE FOR WORD “TERMS”

x	y	i	j	features
Are->terms	SBJ	1	2	……
Not->terms	null	1	3	……
disclose->terms	null	1	4	……
.->terms	null	1	5	……

C. Training instances selection

For a sentence with N words, there will be N^2 instances. In the training procedure, if the ME model computes the entire instances of all the sentences simultaneously the N^2 constituent instances will lead to the memory problem. Therefore, a training instances selection method is essential. We design a statistical method to filter instances which includes three steps: (1) All the arcs in the training data are converted from word pairs into POS tag pairs. (2) For each POS tag pair, the distribution of dependency range (i-j) is counted. (3) Selecting the dependency range which covers 99% instances for each POS tag pair.

TABLE IV.
PART OF THE POS TAG PAIR AND THE SELECTED RANGE"

POS tag pair		Dependency range	
DT	NN	0	3
DT	NNS	0	3
VBD	VBZ	-37	21
.	VBD	-35	1
LS	VB	-8	5
NN	VB	-6	0
……			

Table 4 shows part of the POS tag pair and the selected dependency range. The training instances drops dramatically when the instances selection method is used. Take POS tag pair (DT, NN) as an example, the dependency range is (0, 3). It means that if w_j with POS tag of DT depend on w_i with POS tag NN then 0< i-j < 3.

IV. EXPERIMENTS AND DISCUSSION

Data set for experiments is the Wsj corpus distributed for CoNLL 2008 shared task Closed Challenge. The training data contains 39279 sentences and the test data contains 1334 sentences. The threshold for extensive chart parse algorithm described in section 2.1 is set to 0.85. The Wsj corpus uses 46 different POS tags which are expressive for syntax usage. However, according to the definition of word pattern and POS pattern, the 46 POS tag will lead to the combinative explosion problem. More seriously, some POS tags occur rarely which will lead to the data sparseness problem. After studying the chiastic of those POS tags, we find that those POS tags can be grouped into 7 basic classes. The classed POS tags are shown in table 5.

TABLE V.
POS TAG CLASS TABLE

New POS class	Original POS tags
Noun	NNP, NN, NNS, NNPS,
Verb	VBN, VB, VBD, VBZ, VBP, VBG, MD
Adj	JJ, JJR, JJS, DT, CD, CC, POS, PRF
Adv	RB, RBR, RBS, RP, UH,
Prep	IN, TO
Pron	PRP, PRP$, PDT, WDT, WP, WRB, WP$, EX
Punc	SYM, LS, HYPH, ``, '', (,),.,:,,$,#

The 7 basic kind POS tags can express the main syntactic function of a sentence. The POS tags are converted into the basic POS tags and the basic POS tags are used to extract word patterns and POS patterns. Finally, 100414 word patterns and 15979 POS patterns are extracted from the training data. The pattern number for different POS tags is shown in Fig. 3.

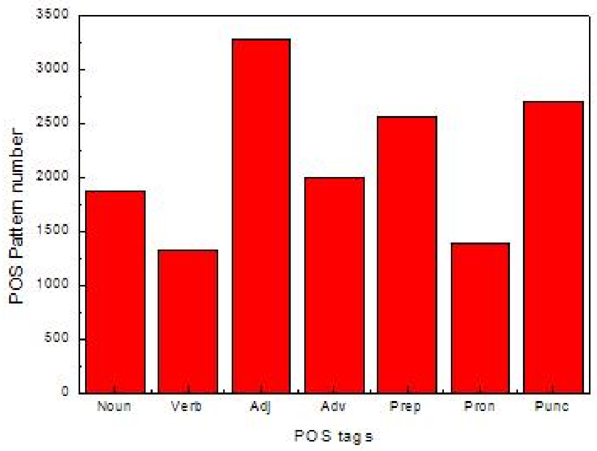


Figure 3. POS Pattern number for different POS tags

Those POS tags with more pattern number such as Preposition, punctuation and adjective tend to be more changeable than those less ones such as Noun and pronoun. A POS pattern contains two parts: POS tag and its elements got from the dependents which has the same arc head. A pattern with more elements is complex and tends to have fewer occurrences. The pattern elements number distribution is shown in Fig. 4. Obviously, most of the patterns contain elements less than 5. That is, the POS patterns are adequate to use.

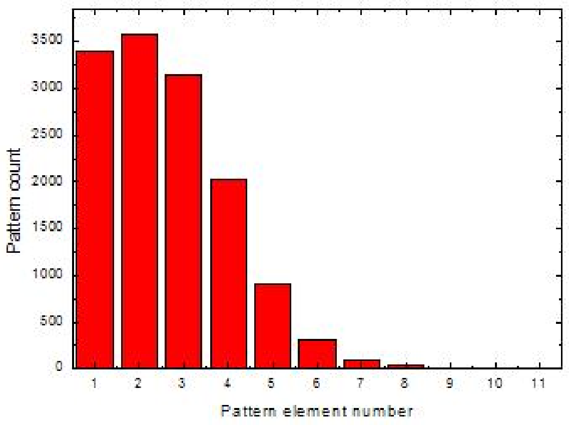


Figure 4. Element number distribution for POS Patterns

To prove the effectiveness of KPDP, we build two other models: pattern model and chart model which only use pattern probability and arc probability, respectively.

Pattern model:

Pattern model uses the constituent parser result proposed in section 2 and re-compute the probability with pattern models but omits the probability from ME model. Let C denotes the constituent parser tree set, the pattern model gives the parser result as:

$$T^* = \arg\max_{T \in C} \prod_{k_i, w \in T} p(k_i \mid w) \qquad (18)$$

$p(k_i \mid w)$ is the pattern probability defined in Eq. (9).

Chart model:

Chart model selects the best tree from the constituent parser tree set directly. Let C denotes the constituent parser tree set, and the chart model gives the parser result as:

$$T^{*} = \arg\max_{T \in C} \prod_{(w_i -> w_j) \in T} p(w_i -> w_j) \qquad (19)$$

A. Experiment results

Two evaluation criteria are used to measure the performance of these three models.

Labeled attachment score:

Proportions of dependency structures which are completely correct.

Arc without label:

Proportions of words and punctuation marks that are assigned the correct arcs without considering labels.

The experiment results are shown in table 6. The proposed KPDP model gets the best performance both in labeled attachment score and arc score. The result is promising and verifies the effectiveness of KPDP. The chart model is equivalent to a native probabilistic model which use the down to up strategy. The proposed KPDP achieve 83.25% for labeled attachment score which is 3.2% higher than Chart model. Although our result is not as good as the best result achieved by the MSTparser [13] which use a large margin strategy and training online, it is the best result comparing with other native probabilistic models. More importantly, the performance verifies that the word knowledge is useful and can be used for dependency parsing.

TABLE VI.
EXPERIMENT RESULS

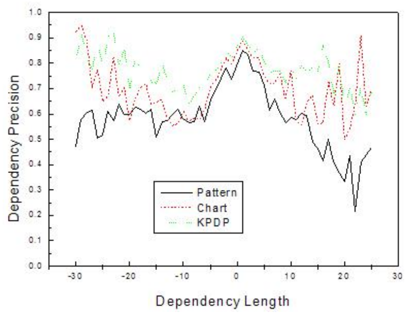
model	Labeled attachment score	Arc without label
Pattern	0.7484	0.800
Chart	0.8045	0.841
KPDP	0.8365	0.87

B. Discussion

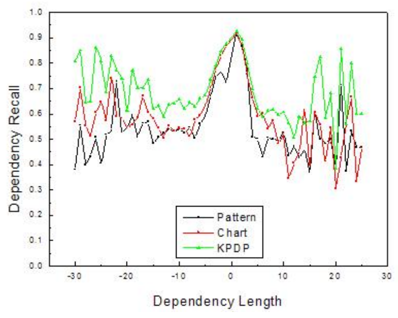
The primary purpose of this section is to learn the characteristics of KPDP by analyzing the performance of these three models. A set of linguistic and structural properties are used to measure the errors of the proposed parsing model.

An important property is accuracy relative to dependency length. The length of a dependency is the distance between two words which form a dependency arc. Let $(w_i -> w_j)$ denotes an arc, the distance is defined as i-j not | i-j|. Therefore we can measure the word dependency on both sides separately. Longer dependencies typically represent modifiers of the root or the main verb in a sentence. Shorter dependencies are often modifiers of nouns such as determiners, adjectives or pronouns modifying their direct neighbors [13].

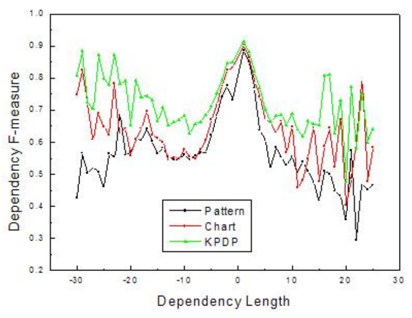
Fig. 5 shows the influence of dependency length on the performance of each model. The dependency distance range is from -30 to 25 which is not balance. This means that words tend to have longer distance dependency on the left side. Statistically, we find that the long range distance is mainly induced by the end mark which depends on the center verb of the sentence on the left side. The short dependencies have a better performance than long dependencies. The pattern model is not sensitive with the dependency distance; therefore its performance curve should be smoother than the Chart model. However, the curve does not prove this assumption at the first glance. When the dependency distance is in the range (-30,-20) and (20, 25) the pattern model performs badly, the errors are caused mainly by mismatching some short patterns. When the dependency distance is in range (-20,-8) and (5, 20) the pattern model curve is smoother than the Chart model. The KPDP model performs better than the other two models in any distance. And the KPDP curve is smoother than the other models, for it takes the advantage of the arc probability and the pattern probability which can express the relationships between arcs.



(a) Precision



(b) Recall



(c) F1

Figure 5. Influence of dependency length on the performance

It is well known that parsing systems tend to have lower accuracies for longer sentences [12]. The well known MSTParser, presented by Ryan McDonald Joakim Nivre 2005, performs badly when the sentence longer than 40 words. Fig. 6 shows that the KPDP model is not sensitive with sentence length as MSTParser. This phenomenon can be explained: KPDP uses word knowledge which is not affected by sentence length. That is why the KPDP curve does not drop sharply as the pattern model does. We also notice that the pattern model perfumes badly when sentence length is less than 5 words. This phenomenon can be interpreted as: The pattern model cannot get efficient patterns when the sentence is short and performs very badly when the sentence is shorter than 5 words. The pattern model also performers bad and unstable when the sentence length is longer than 40 words. It may be induced by the confliction of patterns. Since the pattern probability is calculated statistically, there is no reasonable way to deal the pattern confliction problem. In conclusion, the KPDP performs very steady and smooth though it drops a little with the increasing of sentence length. This fact also proves the superiority of the KPDP.

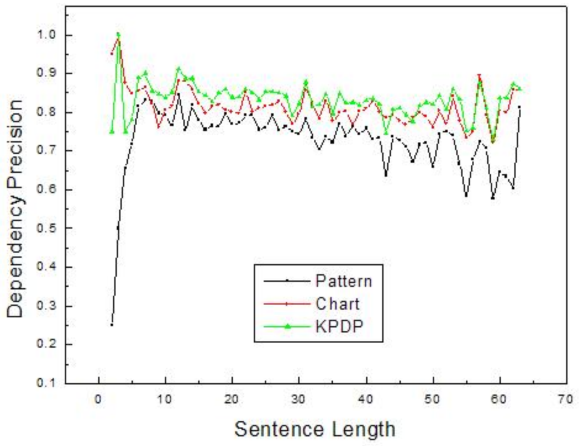
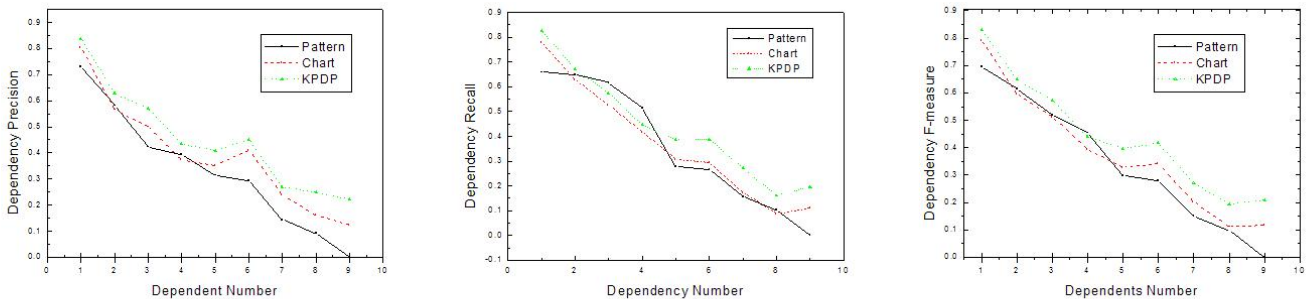


Figure 6. Influence of sentence length on the performance

Sometimes, there are more than one word depending on a specific word such as verb and noun. The words having the same number of dependents or modifiers are grouped into same siblings and the performance of them is shown in Fig. 7. The pattern model has good performance at words with 1-4 dependents. However its performance drops sharply with the increase of dependents number for the data sparseness problem. The pattern model tends to select the patterns with less dependents and this can explain the recall of pattern model which is higher than the other two models when the dependent number is in range 3-5. The KPDP performs better than the other two models all the time. Furthermore, KPDP curve is better obviously than the other two models when the dependent number is more than 5. That is, the KPDP is steady for complex word with more dependent.

Since Pattern model has no arc probability, it uses the arc probability of Chart model instead. The arcs in real dependency tree getting low probability are mainly due to distance reason. Then arc probabilities decrease when the dependency distances increase. Fig. 8 shows the performance curves with the influence of arc probability. Since patterns are not sensitive to dependency distance, it has a smoother performance than Chart model. These three models all reach the best performance at the highest arc probability.



(a) Precision (b) Recall (c) F1

Figure 7. Influence of dependent number on the performance

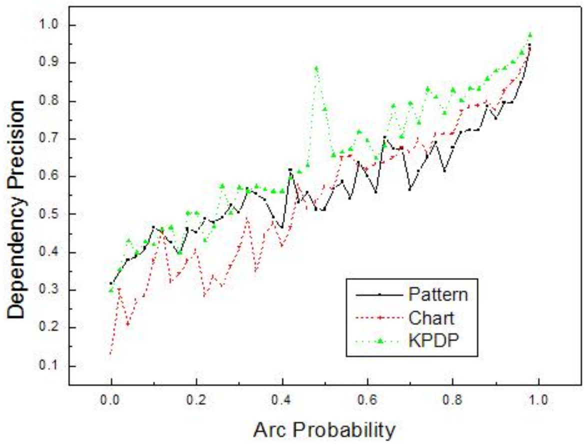


Figure 8. Influence of arc probability on the performance

Fig. 9 shows the performance of these three models for different POS tags and Fig. 10 is the distribution of different POS tags.

These two figures measure labeled dependency accuracy relative to the POS of the dependent word. KPDP model gets the best performance at all the POS tags. This can lead into the results that word knowledge is useful for words with any POS tags. For Noun, Adj (Adjective), Adv (Adverbial), Prep (Preposition), these three models are nearly undistinguishable. However, For Verb, Pron (Pronoun) and Punc (Punctuation), Chart model is a little more precise than pattern model and KPDP is obviously better than the other two models.

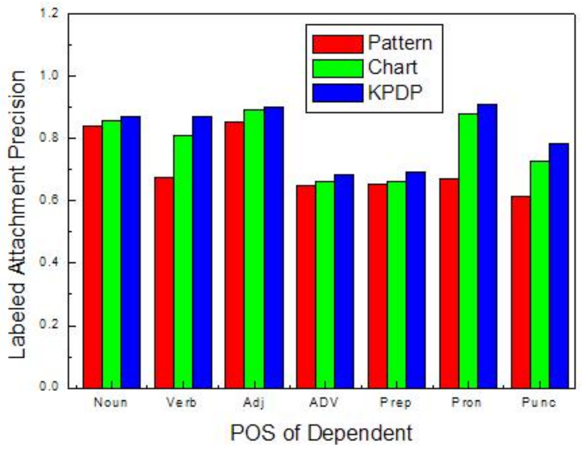


Figure 9. Influence of dependent POS on the performance

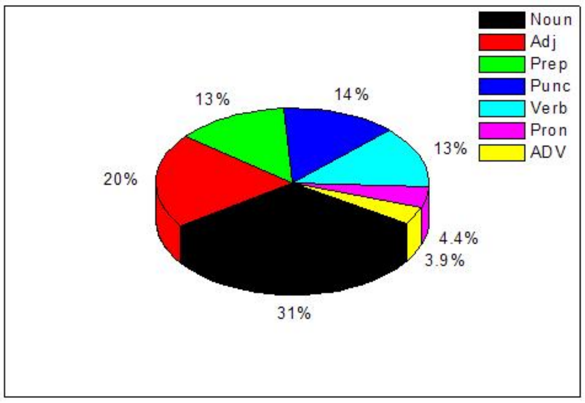


Figure 10. POS tag distribution

V. CONCLUSION

We have presented a new framework KPDP for parsing dependency trees based on word knowledge and probabilistic models. A major advantage of KPDP is its ability to handle the relationships between arcs in a parser tree by using word knowledge which is represented by some patterns. Experiment results are encouraging but not outstanding: the KPDP outperforms the native probabilistic models but not as good as state-of-the-art result achieved by the MSTparser [13] which use a large margin strategy and training online. Furthermore, the error analyzing procedure proves the assumption that KPDP is not as sensitive as native probabilistic models with sentence length and dependency length. And more importantly, the word knowledge, represented as word pattern and POS patterns, is effective for dependency parsing. To integrate work knowledge into parsing procedure, we define an extensive chart parsing algorithm which can filter the exponential constituent parse trees into a set.

ACKNOWLEDGMENT

This work has been mainly supported by the Natural Science Foundation of China (No. 60703015) and the National 863 Program of China (No. 2007AA01Z194).

REFERENCES

[1] Lucien Tesnière. Éléments de syntaxe structurale, Klincksieck, Paris 1959.

[2] Lilja Øvrelid. Linguistic features in data-driven dependency parsing. Proceedings of the 12th Conference on Computational Natural Language Learning, 2008,pp, 25–32

[3] Qin Iris Wang. Learning Structured Classifiers for Statistical Dependency Parsing. Proceedings of the NAACL-HLT, Doctoral Consortium, 2007,pp,5–8

[4] Daisuke Kawahara, Kiyotaka Uchimoto. Minimally Lexicalized Dependency Parsing. Proceedings of the ACL Demo and Poster Sessions,2007, pp,205–208

[5] Eisner, Jason M. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING-96: Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, pp, 340–345.

[6] Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector machines. In *IWPT-2003: 8th International Workshop on Parsing Technology*, pp, 195–206.

[7] Ryan McDonald, Fernando Pereira, Kiril Ribarov, Non-projective Dependency Parsing using Spanning Tree Algorithms". HLT-EMNLP 2005,pp, 523-530

[8] Ryan McDonald, Fernando Pereira. " Online Learning of Approximate Dependency Parsing Algorithms". EACL 2006, pp, 81-88

[9] Fillmore, Charles J. (1968) "The Case for Case". In Bach and Harms (Ed.): *Universals in Linguistic Theory*. New York: Holt, Rinehart, and Winston, pp, 1-88.

[10] Kun Yu, Daisuke Kawahara,Sadao Kurohashi. Chinese Dependency Parsing with Large Scale Automatically Constructed Case Structures. Proceedings of the 22nd International Conference on Computational Linguistics. 2008,pp, 1049–1056

[11] Adam Berger, Stephen Della Pietra, Vincent Della Pietra. "A Maximum Entropy Approach to Natural Language Processing". Computational Linguistics.1996, pp, 39-71

[12] Eugene Charniak. A Maximum-Entropy-Inspired Parser. Proceedings of the first conference on North American. 2000, pp, 132-139

[13] Ryan McDonald, Joakim Nivre. Characterizing the Errors of Data-Driven Dependency Parsing Models. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. 20007, pp, 122–131.