Stochastic Dependency Parsing Based on A* Admissible Search

Bor-shen Lin

National Taiwan University of Science and Technology / No. 43, Keelung Road, Section 4, Taipei, Taiwan

bslin@cs.ntust.edu.tw

Abstract

Dependency parsing has gained attention in natural language understanding because the representation of dependency tree is simple, compact and direct such that robust partial understanding and task portability can be achieved more easily. However, many dependency parsers make hard decisions with local information while selecting among the next parse states. As a consequence, though the obtained dependency trees are good in some sense, the N-best output is not guaranteed to be globally optimal in general.

In this paper, a stochastic dependency parsing scheme based on A* admissible search is formally presented. By well representing the parse state and appropriately designing the cost and heuristic functions, dependency parsing can be modeled as an A* search problem, and solved with a generic algorithm of state space search. When evaluated on the Chinese Tree Bank, this parser can obtain 85.99% dependency accuracy at 68.39% sentence accuracy, and 14.62% node ratio for dynamic heuristic. This parser can output N-best dependency trees, and integrate the semantic processing into the search process easily.

1 Introduction

Constituency grammar has long been the main way for describing the sentence structure of natural language for decades. The phrase structure of sentences can be analyzed by such parsing algorithms as Earley or CYK algorithms (Allen, 1995; Jurafsky and Martin 2001). To parse sentences with constituency grammar, a set of grammar rules written in linguistics is indispensable, while a corpus of tree bank annotated manually is also necessary if stochastic parsing scheme is adopted. In addition, the many non-terminal or phrasal nodes make it sophisticated to further interpret or disambiguate the parse trees since deep linguistic knowledge is often required. All these factors make language understanding very difficult, labor-intensive and not easy to be ported to various tasks.

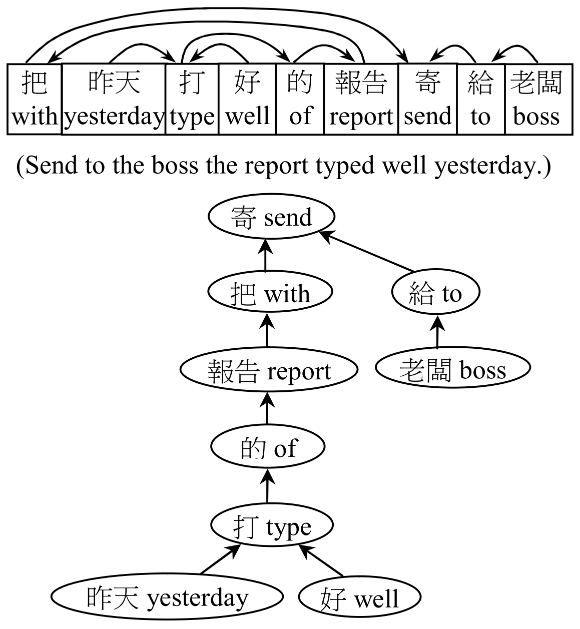

Figure 1. An example of dependency parsing with unlabeled dependency tree.

Dependency grammar, on the other hand, describes the syntactic and semantic relationships among lexical terms directly with binary head-modifier links. Such representation is very simple, compact, direct, and therefore helpful for simplifying the process of language understanding and increasing task portability. Figure 1 shows an example of dependency parsing for the Chinese sentence "把昨天打好的報告寄給老闆" (meaning "send to the boss the report typed well yesterday"). According to the binary head-modifier links, it is pretty easy to transform the dependency tree into the predicate "send(to:boss,object:report(typed(yesterday,well)))" for further interpretation or interaction, since the semantic gap between them is slight and the mappings are thus quite direct. Furthermore, robust partial understanding can be achieved easily when full parse is not obtainable, and measured precisely by simply counting the correct attachments on the dependency tree (Ohno et al., 2004). All these will not be so simple provided that conventional constituency grammar is used.

In the dependency parsing paradigm, several deterministic, stochastic or machine-learning-based parsing algorithms have been proposed (Eisner, 1996; Covington 2001; Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre 2003; Nivre and Scholz, 2004; Chen et al., 2005). Many of them make hard decisions with local information while selecting among next parse states. As a consequence, though the obtained dependency trees are good in some sense, the N-best output is not guaranteed to be globally optimal in general (Klein and Manning, 2003).

On the other hand, A* search that guarantees optimality has been applied to many areas including AI. Klein and Manning (2003) proposed to use A* search in PCFG parsing. Dienes et al. depicted primarily the idea of applying A* search to dependency parsing, but there is not yet formal evaluation results and discussions based on the literatures we have (Dienes et al., 2003).

In this paper, a stochastic dependency parsing scheme based on A* admissible search is formally presented. By well representing the parse state and appropriately designing the cost and heuristic functions, dependency parsing can be modeled as an A* search problem, and solved with a generic algorithm of state space search. This parser has been tested on the Chinese Tree Bank (Chen et al., 1999), and 85.99% dependency accuracy at 68.39% sentence accuracy can be obtained. Among three types of proposed admissible heuristics, dynamic heuristic can achieve the highest efficiency with node ratio 14.62%. This parser can output N-best dependency trees, and integrate the semantic processing into the search process easily.

2 Fundamentals of A* Search

Search is an important issue in AI area, since the solutions of many problems could be automated if they could be modeled as search problems on state space (Russell and Norvig, 2003). A well known example is the traveling salesperson problem, as shown in the example of Figure 2. In this section basic constituents of A* search will be depicted.

2.1 State Representation

State representation is the first step for modeling the problem domain. It involves not only designing the data structure of search state but indicating the way a state can be uniquely identified. This is definitely not a trivial issue, and depends on the how the problem is defined. In Figure 2, for example, search state cannot be represented simply with the current city, because traveling salesperson problem requires every city has to be visited exactly once. The two nodes of city E on level 2 in Figure 2, with paths A-B-E and A-C-E respectively, are therefore regarded as different search states, and generate their successor states individually. The node E with the path A-C-E, for example, can generate successor B, but the one with the path A-B-E cannot due to reentry of city B. In other words, the search state here is the path, including all cities visited so far, instead of the current city. However, if the problem is changed as finding the shortest path from city A to city F, the two paths A-B-E and A-C-E can then be merged into a single node of city E with shorter path tracked. In such case, the search state will then be the current city instead of the path.

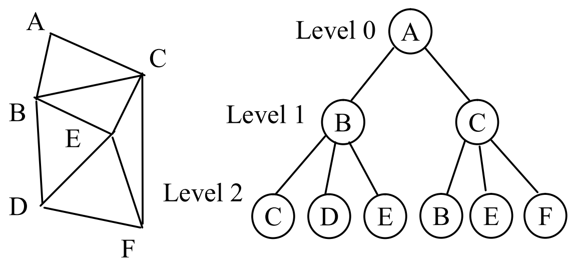

Figure 2. An example of state space search for traveling salesperson problem from initial city A.

In addition, for state representation three issues need to be further addressed.

- Initial state: what the initial state is.
- State transition: how successor states are generated from the current state.
- Goal state: to judge whether the goal state is achieved or not.

In the above example in Figure 2, the initial state is the path containing city A only. The state transition is to visit next cities from the current city under the constraint of no reentry. The goal states are any paths that depart from city A and pass every city exactly once.

2.2 State Space Search

With the state well represented, two data structures are utilized to guide the search.

- An open list (a priority queue, denoted as *open* in Figure 3) used for tracking those states not yet spanned.
- A closed list (denoted as *closed* in Figure 3) used for tracking those states visited so far to avoid the reentry of the same states.

Then, a generic algorithm for state space search, with pseudo codes in object-oriented style shown in Figure 3, is performed to find the goal states. The initial state is first inserted into the queue, and an iterative search procedure (denoted as *search()* in Figure 3) is then called. For each iteration in the search procedure, the search state is popped from the queue and inspected one by one. If it is the goal state, the procedure terminates and returns the goal state. Otherwise the successors of the current state are generated, and inserted into the queue individually according to the priority provided not yet visited. The search procedure could be called multiple times if more than one goal states are desired.

Note that the algorithm in Figure 3 is generic enough to be adapted for various search strategies, including depth first search, breadth first search, best first search, algorithm A, algorithm A* and so on, by simply providing different evaluation function *f(n)* of search state *n* for prioritizing the queue. For depth first search, for example, those states with the highest depth will have higher priority and be inserted at the front of the queue, while for breadth first search at the back.

```
open.add(initial);
goal = search();
search() {
    while(true) {
        state = open.pop();
        if(state.isGoal()) return state;
        successors = state.getSuccessors();
        for(successor in successors) {
            if(!closed.contains(successor)) {
                closed.add(successor);
                open.add(successor);
            }
        }
    }
}
```

Figure 3. Generic algorithm for state space search.

2.3 Optimality and Efficiency

If the evaluation function *f(n)* satisfies

$f(n) = g(n) + h(n)$,

where $g(n)$ is the real cost from the initial state to the current state n and $h(n)$ is the heuristic estimate of the cost from the current state n to the goal state, such type of algorithm is called algorithm A. If further, the constraint of admissibility for $h(n)$ holds, i.e.,

$h(n) \leqq h^*(n)$,

where $h^*(n)$ is the true minimum cost from current state n to the goal state, then optimality can be guaranteed, or equivalently, the goal state obtained first will give minimum cost. Such type of algorithm is called algorithm A*. It can be proved that for algorithm A*, the closer the heuristic $h(n)$ is to the true minimum cost, $h^*(n)$, the higher the search efficiency is.

3 A*-based Dependency Parser

In this section, how dependency parsing is modeled as an A* search problem will be depicted in detail.

3.1 Formulation

First, let $\boldsymbol{W} = \{W_1, W_2, ..., W_n\}$ denote the word list (sentence) to be parsed, where W_i denotes the i-th word in the list. And, each word W_i is expressed in the form,

$$W = (w, t) \qquad (1)$$

where w is the lexical term and t is the part-of-speech tag. A dependency database can first be built by extracting the dependency links among words from a corpus of tree bank. Given the dependency database, a directed dependency graph ***G*** indicating valid links for the word list ***W*** can be constructed, as shown in the example of Figure 4. The direction of the link here indicates the direction of modification. Link $W_3 \rightarrow W_2$ in Figure 4, for example, means W_3 can modify (or be attached to) W_2. The dependency graph ***G*** will be used for directing the state transition during search.

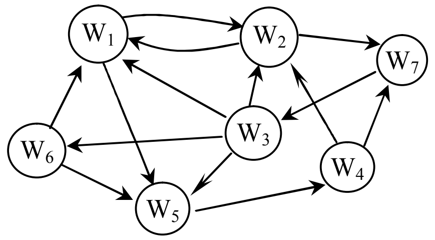

Figure 4. Example dependency graph.

3.2 Representation of Parse State

Since the goal of dependency parsing is to find the dependency tree, the search state can therefore be represented as the dependency tree constructed so far (denoted as ***T***). Besides, a set containing those words not yet attached to the dependency tree (denoted as ***C***, meaning "to be consumed") can be generated dynamically by excluding those words on the dependency tree ***T*** from word list ***W***. The three issues depicted in Section 2.1 are then addressed as follows.

- Initial state: the dependency tree ***T*** is empty and the set ***C*** contains all words in list ***W***.
- State transition: a word is extracted from ***C*** and attached onto some node on the dependency tree ***T***, and a successor state ***T'*** and ***C'*** is then generated. Whether an attachment is valid or not is determined according to the dependency graph ***G***, under the constraints of uniqueness (any word has at most one head) and projectivity[1] (Covington 2001, Nivre 2003).
- Goal state: the set ***C*** is empty while all words in ***W*** are attached onto the dependency tree ***T***.

With the parse state represented well, the generic algorithm for state space search depicted in Section 2.2 can then be performed to find the N-best dependency trees. A partial search space based on the dependency graph ***G*** in Figure 4 is displayed in Figure 5, in which R denotes the root node of dependency tree. As shown in Figure 5, for each search state (denoted by the ovals enclosed with double-lines), only a link in form of $W_j \rightarrow R$ or $W_j \rightarrow W_i$ is actually tracked Through tracing the search tree back from the current state, all links can be obtained, and the overall dependency tree ***T*** can be constructed accordingly. For the search state $W_3 \rightarrow W_2$ in Figure 5, for example, the links $W_1 \rightarrow R$, $W_2 \rightarrow W_1$ and $W_3 \rightarrow W_2$ are obtained through back tracing, which can then construct the dependency tree, $W_3 \rightarrow W_2 \rightarrow W_1 \rightarrow R$, as can be seen on the left-hand side of Figure 5. This is similar to the case of traveling salesperson problem in Figure 2, in which only the current city is tracked for each state, but the path that identifies the search state uniquely can be obtained by back tracing.

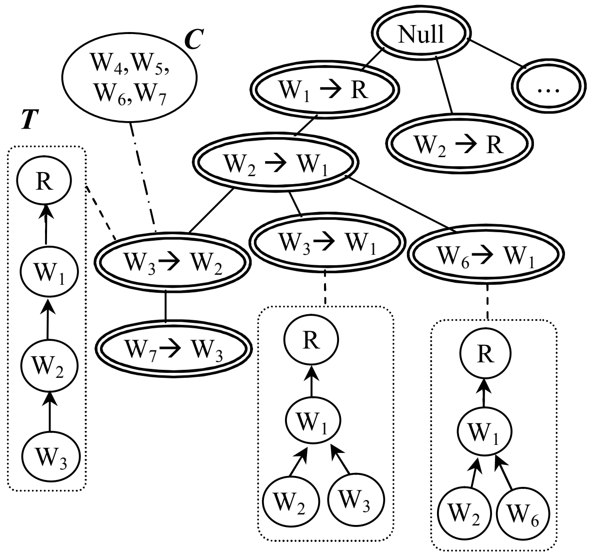

Figure 5. A partial search space where some search states are associated with their dependency trees with dashed lines.

[1] The projective constraint for new attachment is imposed by those attachments already on the dependency tree. Each attachment already on the tree forms a non-crossing region between its head and modifier to constrain new attachments for those words located within that region.

3.3 Cost Function

For given word list $\boldsymbol{W}$, the dependency tree $\boldsymbol{T}$ with the highest probability $P(\boldsymbol{T})$ is desired, where

$$P(\boldsymbol{T}) = \{\Pi P(W_l \rightarrow R)\} \cdot \{\Pi P(W_j \rightarrow W_i)\} \qquad (2)$$

The first term corresponds to those attachments on the root node of dependency tree (i.e. headless words), while the second term corresponds to the other attachments. In A* search, the optimal goal state obtained is guaranteed to give the minimal cost. So, if the cost function, $g(n)$, is defined as the minus logarithm of $P(\boldsymbol{T})$,

$$\begin{aligned} g(n) &\equiv -log\,(P(\boldsymbol{T})) \\ &= \Sigma\,(-log(P(W_l \rightarrow R))) + \Sigma\,(-log(P(W_j \rightarrow W_i))) \\ &\equiv \Sigma\, step\text{-}cost \qquad (3) \end{aligned}$$

then the A* search algorithm that minimizes the cost $g(n)$ will eventually maximizes the probability of the dependency tree, $P(\boldsymbol{T})$. Here the minus logarithms of the probabilities in Equation (3), $-log(P(W_l \rightarrow R))$ and $-log(P(W_j \rightarrow W_i))$, can be regarded as the step cost for each attachment (or link) accumulated during search. Furthermore, since each word is expressed with a lexical term and a part-of-speech tag as shown in Equation (1), the probability for each link can be depicted as,

$$P(W_l \rightarrow R) = P(W_l \mid R) = P(t_l|R) \cdot P(w_l|t_l) \qquad (4)$$

$$P(W_j \rightarrow W_i) = P(W_j \mid W_i) = P(t_j|t_i) \cdot P(w_j|t_j) \qquad (5)$$

assuming the word list $\boldsymbol{W}$ is generated by the Baysian networks in Figure 6(a) and 6(b) respectively. Note that here the probability for a link $P(W_j \rightarrow W_i)$ involves not only the lexical terms w_j and w_i but also the part-of-speech tags t_j and t_i, and is denoted as link bigram. Such formulation can be generalized to high order link n-grams. Figure 6(c), for example, displays the Baysian network for the link $W_k \rightarrow W_j$ conditioned on $W_j \rightarrow W_i$, based on which the link trigram is defined as

$$\begin{aligned} P(W_k \rightarrow W_j \mid W_j \rightarrow W_i) &= P(W_k \mid W_i, W_j) \\ &= P(t_k \mid t_i, t_j) \cdot P(w_k \mid t_k). \qquad (6) \end{aligned}$$

When comparing Figure 6(c) with Figure 6(d), the Baysian networks for link trigram $P(W_k \mid W_i, W_j)$ and conventional linear trigram $P(W_{i+2} \mid W_i, W_{i+1})$ respectively, it could be found that link n-gram is flexible for long-distance stochastic dependencies, though the two topologies look similar. Undoubtedly, the Baysian networks in Figure 6 appear too simple to model the real statistics precisely. In the link $W_j \rightarrow W_i$, for example, W_j might depend on not only its parent W_i but the children of W_i (Eisner, 1996). Also, the direction ($sgn(i-j)$) and the distance ($|i-j|$) of modification between W_i and W_j might be important (Ohno et al., 2004). All these factors could be taken into account by simply including the minus logarithms of the corresponding probabilities into the step cost.

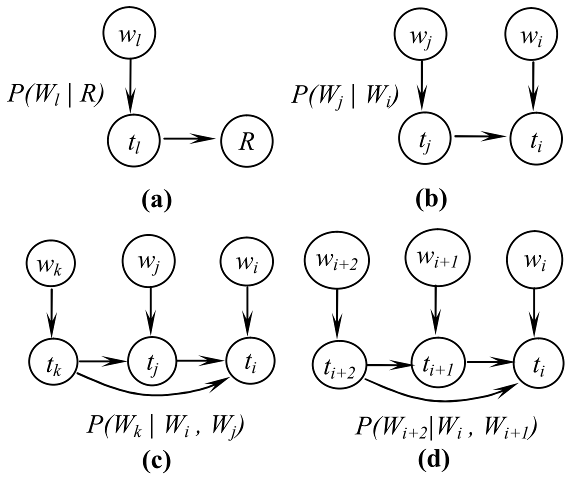

Figure 6. Baysian networks of (a) link unigram (b) link bigram (c) link trigram (d) linear trigram.

3.4 Heuristic Function

In A* search, the evaluation function $f(n)$ consists of $g(n)$, the real cost from the initial state to the current state n, and $h(n)$, the cost estimated heuristically from the current state n to the goal state. Now for dependency parsing, $g(n)$ defined in Equation (3) is the accumulated cost for those attachments on current dependency tree $\boldsymbol{T}$, while $h(n)$ is the predicted cost of the attachments for the words in $\boldsymbol{C}$ which have not yet been attached. Since $h(n) \leqq h^*(n)$ has to hold for admissibility, it is necessary to estimate $h(n)$ conservatively enough so as not to exceed the true minimum cost $h^*(n)$. To achieve higher search efficiency, however, it is preferred to estimate $h(n)$ more aggressively such that $h(n)$ can be as close to $h^*(n)$ as possible. Therefore, in this paper, $h(n)$ is estimated with *the minimum of minus logarithms of link n-grams* for each word in $\boldsymbol{C}$, with different levels of constraints described below.

- Global heuristic: the minimum is static, and can be calculated in advance before parsing any sentence by considering all link n-grams for each word.
- Local heuristic: the minimum is calculated according to the dependency graph ***G*** of current parsing sentence by considering all possible link n-grams with respect to ***G***.
- Dynamic heuristic: the minimum is calculated dynamically according to current dependency tree ***T*** during parsing by considering only projective link n-grams with respect to ***G*** for current dependency tree ***T***.

By virtue of taking the minimum of minus logarithms of link n-grams, admissibility can be guaranteed for these heuristics. The latter with stricter constraints on finding the minimum can give more precise estimate of cost (with higher $h(n)$) than the former, and is thus expected to be more efficient, as will be discussed in the next section.

4 Experiments

The stochastic dependency parsing scheme proposed in this paper has been first tested on the Chinese Tree Bank 3.0 licensed by Academia Sinica, Taiwan, with six sets of data collected from different sources (Chen et al., 1999). Head-modifier links (with lexical term and part-of-speech tag) can be extracted from the tree bank since it is produced by a head-driven parser (Chen, 1996). In our experiments, One set, named as *oral.check*, containing 4156 sentences manually transcribed from dialogue database is used to train the dependency database, link statistics including conditional probabilities, link unigrams and bigrams as shown in Equation (4) and (5), and so on. Another set, named as *ev.check*, containing 1797 sentences in text books of elementary school is used to test the A*-based dependency parser. Note here the training corpus and testing corpus are of different domains, and each word in test sentences is transcribed into (w, t) format with lexical term w and associated part-of-speech tag t.

4.1 Coverage Rate

The number of occurrences and the coverage rate for the conditional probability $P(w_j|t_j)$, link unigram $P(t_j)$ and link bigram $P(t_j|t_i)$ respectively are shown in Table 1. As can be seen in this table, the coverage rate for $P(w_j|t_j)$ is as low as 50.8%, since the training and testing domains are quite different. The coverage rates for link unigram and bigram, however, can be up to 94.06% and 84.88% respectively. This implies that, the link probabilities can be estimated more appropriately and contribute more in finding the dependency trees. To handle the issue of data sparseness, in the following experiments a simple n-gram backoff mechanism is utilized for smoothing.

No. of occurrences	$P(w_j\|t_j)$	$P(t_j)$	$P(t_j\|t_i)$
Training corpus	8137	120	3383
Testing corpus	2791	101	1468
Overlaps	1418	95	1246
Coverage rate	50.80%	94.06%	84.88%

Table 1. Coverage rates for link statistics.

4.2 Parsing Accuracy

The experiment settings for dependency parsing are depicted as below. The first, denoted as BASE, is the baseline setting, while the others describe the search conditions applied to the baseline incrementally. The heuristic $h(n)$ used here is the dynamic heuristic.

- BASE: baseline with the cost defined in Equation (3), (4) and (5).
- RP: root penalty for every root attachment $W_l \rightarrow R$ is included into the step cost.
- DIR: the statistics for the direction of modification, $P(D|W_j \rightarrow W_i)=P(D|t_i, t_j)$, is included into the step cost where $D \equiv sgn(j-i)$.
- NA: the statistics for the number of attachments (or valence), $P(N_i|W_i)=P(N_i|t_i)$, is included into the step cost and updated incrementally for every new attachment onto W_i. N_i is the current number of words attached to W_i.
- REJ: the obtained dependency trees are inspected one by one, and rejected if any of the conditions occurs: (a) the modifiers for conjunction word (e.g. "和", meaning "and") belong to different part-of-speech categories (incorrect meaning) (b) illegal use of "的" (meaning "of") as leaf node (incomplete meaning).

The baseline setting with Equation (3), (4) and (5) is based on the Baysian networks depicted in Figure 6(a) and 6(b). Finer statistics could be applied in the settings RP, DIR and NA. The setting RP, for example, can restrain the number of headless words. The setting DIR can take into account the cost due to the direction of modification, since it in fact matters[2], but the link probability $P(W_j \rightarrow W_i)$ in Equation (5) cannot differentiate between the two directions. The distance of modification, |j-i|, might matter too, but is not considered here due to quite limited amount of training data. The setting NA can include the cost due to the number of attachments. Verbs, for example, often require more attachments, and may produce lower cost for higher number of attachments. Here for simplification, the statistics of $P(N_i|t_i)$ are gathered only for N_i = 0,1,2, and >=3, respectively. In addition to using finer statistics, it is also feasible to use semantic or lexical rules to reject the dependency trees with incorrect or incomplete meanings. In the setting REJ, two rules are utilized. One is, the conjunction word should have modifiers in the same part-of-speech category, while the other is, the word "的" must have at least one modifier.

		No. of Accurate Sentences	SA	DA
Cross Domain	BASE	867	48.25%	77.42%
	+ RP	1039	57.82%	80.74%
	+ DIR	1102	61.32%	82.72%
	+ NA	1211	67.39%	85.21%
	+ REJ	1229	68.39%	85.99%
Within Domain	BASE	1109	61.71%	85.93%
	+ RP	1314	73.12%	89.91%
	+ DIR	1340	74.57%	90.66%
	+ NA	1476	82.16%	92.81%
	+ REJ	1484	82.58%	93.20%

Table 2. Parsing accuracy for various settings.

The parsing performance can be measured with sentence accuracy (SA) and dependency accuracy (DA). Table 2 shows the experimental results for the above settings. The results for within-domain test by using the set *ev.check* for both training and testing are also listed here for comparison. It can be found in this table that, the performance of cross-domain test is not so good for the baseline setting, but can be persistently improved when finer statistics are applied. Rejection of incorrect or incomplete dependency trees is also helpful (REJ), though very few semantic or lexical constraints are utilized here. When the constraints of all settings are applied, 85.99% dependency accuracy can be obtained at 68.39% sentence accuracy.

4.3 N-best Output

Note that due to data sparseness and the limitation of simplified Baysian networks, some parsing errors are intrinsic and difficult to avoid. Figure 7 shows the parsing result for a clause "吃晚飯的時候" (meaning "at the time for eating dinner"). The incorrect parsing result on the right-hand side (meaning "to eat the time of dinner") is syntactically correct and inevitable in fact, since the link probabilities ($P(t_j)$ or $P(t_j|t_i)$) dominate over the conditional probabilities ($P(w_j|t_j)$), as illustrated in Section 4.1. Such problem could possibly be alleviated to some extent if deep semantic constraints (e.g. a transitive verb may require a subject and an object) or lexical constraints (e.g. some adjectives may modify only specific nouns) could be utilized for rejection while reprocessing the N-best output.

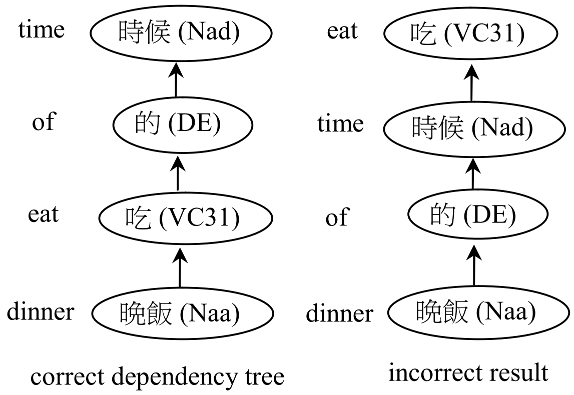

Figure 7. The parsing result for the clause "吃晚飯的時候" (time for eating dinner).

Table 3 shows the experimental results of N-best output for the setting REJ. In Table 3 it can be seen that higher sentence accuracy, 80.08%, for top-5 output can be achieved, which implies large space for improvement in N-best processing.

	Top 1	Top 2	Top 3	Top 4	Top 5
SA	68.39%	75.24%	78.02%	79.41%	80.08%

Table 3. Sentence accuracy for N-best output.

[2] In the Chinese phrase "在(in)…前面(front)", for example, "在" is the head while "前面" is the modifier, and "前面" always modifies "在" backwards.

4.4 Search Efficiency

The search efficiency for each test sentence can be measured heuristically by the order of complexity, defined as

$$C = log_n N \qquad (7)$$

where n is number of words in the test sentence and N is the total number of the search nodes for that sentence. C_{ave} is then used to denote the average complexity over all test sentences. In addition, N_{all} is the total number of search nodes for all test sentences, and can produce the node ratio R_N when normalized. The experimental results for different heuristics depicted in Section 3.4 are shown in Table 4. As can be observed in this table, much higher search efficiency can be obtained for more precise heuristic estimate, but with compatible top 1 sentence accuracy. For dynamic heuristic with real-time estimate of the cost according to the current dependency tree, the highest efficiency can be obtained at 2.38 average complexity and 14.62% node ratio, respectively.

Heuristic	SA	C_{ave}	N_{all}	R_N
None	68.84%	2.91	4748268	100%
Global	68.34%	2.82	3417766	66.29%
Local	68.50%	2.39	841716	17.73%
Dynamic	68.39%	2.38	694193	14.62%

Table 4. Search efficiency for different heuristics.

5 Conclusions

We have presented a stochastic dependency parsing scheme based on A* admissible search, and verified its parsing accuracy and search efficiency on the Chinese Tree Bank 3.0. 85.99% dependency accuracy at 68.39% sentence accuracy can be obtained under cross-domain test. Among three types of admissible heuristics proposed, dynamic heuristic can achieve the highest efficiency with node ratio 14.62%. This parser can output N-best dependency trees, and reprocess them flexibly with more semantic constraints so as to achieve higher parsing accuracy. This parsing scheme is the basis for natural language understanding in our research project on dialogue systems for mission delegation tasks. We plan to perform comparative studies with other non-stochastic approaches, and evaluate our approach on the shared task of dependency parsing.

References

Allen James, 1995. *Natural Language Understanding*, The Benjamin/Cummings Publishing Company.

Chen K. J. 1996. *A Model for Robust Chinese Parser*, Computational Linguistics and Chinese Language Processing, Vol. 1, no. 1, pp. 183-205.

Chen K. J., et al. 1999. *The CKIP Chinese Treebank: Guidelines for Annotation*, ATALA Workshop – Treebanks, Paris, pp. 85-96.

Chen Yuchang, Asahara Masayuki and Matsumoto Yuji. 2005. Machine Learning-based Dependency Analyzer for Chinese, ICCC-2005.

Covington Michael A. 2001. *A Fundamental Algorithm for Dependency Parsing*, Proc. of the 39th Annual ACM Southeast Conference, pp. 95-102.

Dienes Peter, Koller Alexander and Kuhlmann Macro. 2003. *Statistical A* Dependency Parsing*, Proc. Workshop on Prospects and Advances in the Syntax/Semantics Interface.

Eisner Jason M. 1996. *Three Probabilistic Models for Dependency Parsing: An Exploration*, Proc. COLING, pp. 340-345.

Jurafsky Daniel and Martin James H. 2001. *Speech and Language Processing*, Prentice Hall, NJ.

Klein Dan and Manning Christopher D. 2003. *A* Parsing: Fast Exact Viterbi Parse Selection*, Proc. NAACL/HLT.

Klein Dan and Manning Christopher D. 2003. *Fast Exact Inference with a Factored Model for Natural Language Parsing*, Proc. Advances in Neural Information Processing Systems.

Kudo Taku and Matsumoto Yuji. 2002. *Japanese Dependency Analysis using Cascaded Chunking*, Proc. CONLL.

Nivre Joakim, 2003, *An Efficient Algorithm for Projective Dependency Parsing*, Proc. International Workshop on Parsing Technologies (IWPT).

Nivre Joakim and Scholz Mario. 2004. *Deterministic Dependency Parsing of English Text*, Proc. COLING.

Ohno Tomohiro, et al, 2004. *Robust Dependency Parsing of Spontaneous Japanese Speech and Its Evaluation*, Proc. ICSLP.

Russell S. and Norvig P. 2003. *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Yamada Hiroyasu and Matsumoto Yuji. 2003. *Statistical Dependency Analysis with Support Vector Machines*, Proc. IWPT