A* decoder for shift-reduce constituent parsing with global discriminative training

Thang Le Quang
SoICT - HUST
lelightwin@gmail.com

Yusuke Miyao
National Informatic Institute
Yusuke@nii.ac.jp

Hiroshi Noji
National Informatic Institute
Noji@nii.ac.jp

Abstract

The feature-based shift-reduce constituent parsers now has become one of the state-of-the-art parsing systems due to their efficiency. But there still remains a problem: most of the popular shift-reduce constituency parsers relied on inexact search which cannot guarantee the exactness. In this paper, we will propose a novel idea which based on A* algorithm to perform an exact search for shift-reduce constituent parsing. An exact search will lead to an optimal model for global training of shift-reduce parser and gives a better performance than inexact search. The final experiments has shown that our parser has outperformed inexact search with the same feature set in terms of F-score and can get a comparable accuracies to the previous state-of-the-art parsers.

1 Introduction

Shift-reduce parsing is one of the most classical method for syntactic parsing. In this approach, a parse tree is considered as a set of shift-reduce actions. In all possible parse trees whose scores are the sum of model scores assigned to its shift-reduce actions, the system would select the candidate having the highest score as a result. The approach of shift-reduce parsers is very successful. The advantage of this method is that it can exploit very rich features with a global decoding which leads to very high accuracies. (Mengqiu and Teruko, 2006) present the first research on shift-reduce parser which can reach to 88% F-score on English WSJ tree bank using SVM approach. In (Yue and Stephen, 2009), they has proposed out the global discriminative training for shift-reduce parsing which can achieve state-of-the-art performance for Chinese constituent parsing with the use of averaged perceptron from (Michael and Brian, 2004). Utilizing and extending the approach of (Yue and Stephen, 2009), (Zhu et al., 2012) has built a fast and accurate constituent shift-reduce parser which even outperforms the previous state-of-the-art CYK-based parses such as Berkeley parser and Stanford parser.

However, there is a problem which is still existed in those shift-reduce parsing system: all of them has to use inexact search to do the work in the decoding process. (Mengqiu and Teruko, 2006) uses greedy search performing a deterministic decoder which must sacrifice a large space of candidate parse trees. (Yue and Stephen, 2009) and (Zhu et al., 2012) use beam search to extend the candidate space but it still cannot guarantee the exactness. We have a hypothesis that inexact search such as beam and greedy method may cause some searching errors and make the performance become worse than exact search. Therefore, in order to investigate the effect of this search errors, we would like to propose a strategy to perform an exact search for shift-reduce parsing by using A* heuristic. We apply our heuristic into the baseline shift-reduce constituent parsing of (Yue and Stephen, 2009) to test our hypothesis. In our understanding, there is no previous research which can achieve the exactness on global discriminative training for shift-reduce parsing. Our final experiments on WSJ dataset has shown that our parser can overcome the previous constituent shift-reduce parsers in terms of F-score and even gives a comparable accu-

racy to the state-of-the-art parsers.

Related works

(Dan and Christopher, 2003a) has proposed the first research on A* parsing for PCFG model. This system has maintained an agenda to store the processed nodes and extend them in a best-first order by using inside scores and an A* heuristic of outside scores. In their system, Klein and Manning summarized the context to calculate the A* heuristic of the outside probability. Therefore, they have to precomputed and stored the scores of all possible context summaries. They have reported that this heuristic can ignore more than 97% of total possible nodes and lead the search reaching to goal very fast. However, A* heuristic is very difficult to be applied into the shift-reduce parsers. There are two main challenges in this approach: the first one is the very large search space caused by their rich features. The second one is that the model scores in shift-reduce parsing are always updated overtime due to the incremental training and cannot be directly precomputed as in (Dan and Christopher, 2003a).

Actually, there are still few parsers which try to perform exact search on shift-reduce parsing. (Liang and Kenji, 2010) uses dynamic programming to reduce the total time complexity to polynomial but they still have to use beam search in the decoding process. (Kenji and Alon, 2006) and (Kai et al., 2013) has presented their ways of how to utilize best first search in shift-reduce parsing. However, these two system are locally trained on the maximum entropy model and their performances are still very far behind from the shift-reduce parsing system which has been trained by global discriminative method such as (Zhu et al., 2012) or (Yue and Stephen, 2009). In our system, we firstly build the best first shift-reduce constituent parsing based on the dynamic algorithm of (Liang and Kenji, 2010), and then apply our proposed A* heuristic to make the exact search in the shift-reduce approach become possible for the first time.

2 Our dynamic Shift-Reduce parser

In order to apply the A* heuristic, we have developed our own shift-reduce constituent parser based on dynamic algorithm. Within our knowledge, this is the first implementation of dynamic algorithm on shift-reduce constituent parsing.

$$\begin{array}{ll}
\text{input:} & w_0 ... w_{n-1} \\
\text{axiom} & 0 : \langle 0, \epsilon \rangle : 0 \\
\text{sh} & \dfrac{l : \langle j, S \rangle : c}{l+1 : \langle j+1, S|w_j \rangle : c + \xi} \quad j < n \\
\mathrm{b}_L(X) & \dfrac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|X(s_1) \rangle : c + \lambda_L} \\
\mathrm{b}_R(X) & \dfrac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|X(s_0) \rangle : c + \lambda_R} \\
\mathrm{u}(X) & \dfrac{l : \langle j, S|s_0 \rangle : c}{l+1 : \langle j, S|X(s_0) \rangle : c + \lambda_U} \\
Fin & \dfrac{2n + \alpha - 1 : \langle n, s_0 \rangle : c}{2n + \alpha : \langle n, \emptyset \rangle : c + \rho}
\end{array}$$

where l is the step, c is the total cost. the shift cost ξ, left/right binary cost λ_L, λ_R, unary cost λ_U, finish cost ρ are:

$$\xi = w \cdot \Phi_{sh}(j, S) \tag{1}$$

$$\lambda_L = w \cdot \Phi_{b_L(X)}(j, S) \tag{2}$$

$$\lambda_R = w \cdot \Phi_{b_R(X)}(j, S) \tag{3}$$

$$\lambda_U = w \cdot \Phi_{u(X)}(j, S) \tag{4}$$

$$\rho = w \cdot \Phi_{Fin}(2n + \alpha, s_0) \tag{5}$$

Figure 1: Deductive system for conventional shift-reduce constituent parser.

2.1 Conventional shift-Reduce Constituent Parsing

Shift-reduce parsing can be considered as a state transition system, which means that it includes a set of states and actions. From a current state, each action will create a new corresponding state. A state consists of two data structures: a stack S of constructed phrases and a queue Q of remain incoming words in the input sentence. As in (Yue and Stephen, 2009), a constituent parser have four types of action:

- SHIFT (denoted sh): pop the front word q_0 from Q and push it onto stack S.
- B-REDUCE-L/R(X) (denoted $\mathrm{b}_{L/R}(X)$): pop the two top nodes s_0 and s_1 from S and use the

Table 1: Averaged perceptron algorithm

Inputs	Training examples (x_i, y_i)
Initialization	$\vec{w}$ = 0
Output	averaged weights $\vec{w}$
Algorithm	For $t = 1...T, i = 1...n$ — Calculate $F(x_i) = argmax_{y \in Beam(x_i)} Score(y)$ — If $(F(x_i) \neq y_i)$ then $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$

binary rule set to combine them into new constituent node with label X, and push X onto *S*. The parameter [L/R] decide the head of compound nodes to be left or right, respectively.

- U-REDUCE(X) (denoted u(X)): pop the top nodes s_0 off *S*, use the unary rule set to combine them into new constituent node with label X, and push X onto stack *S*.
- FINISH (denoted Fin): pop the root node off *S* and end the shift-reduce parsing.

Figure 1 illustrates the deductive system of these four shift-reduce actions [1]. Similar to (Yue and Stephen, 2009), our conventional parser was trained by global averaged perceptron which has been shown in table 1.

2.2 Baseline feature template

We adopted the baseline features from (Yue and Stephen, 2009) which are shown in table 2. In this template, s_i represents the i^{th} item on the top of the stack S and q_i denotes the i^{th} item in the queue Q of incoming words. $s_i l$, $s_i r$ and $s_i u$ are respectively the left, right, and unary child of s_i. There are three components which are used for each item X:

- $X.t$ - denotes the PoS of head lexical of X.
- $X.w$ - denotes the head lexical of X.
- $X.c$ - denotes the constituent tag of X.

2.3 Our modified shift-reduce actions

In conventional shift-reduce constituent parsing, the parse trees for an input sentence may have different number of shift-reduce actions due to the unary reduce actions. As in figure 1, the total number of actions is $2n + \alpha$ with α is unknown. This problem may reduce the comparability between two arbitrary states and lead to unstable performance (Yue and Stephen, 2009).

In (Zhu et al., 2012), they use a padding method which adds the IDLE action in order to extend the sooner completed states. The IDLE action does not change the current state itself, just to keep the number of actions consistency. However, this method creates the redundancy which may increase the number of actions to $4n$ (n is the length of input sentence) and can only guarantee the local consistency in the beam width. With our system, we the following shift-reduce actions to solve the unary rules problem:

- SHIFT: the conventional SHIFT action.
- B-REDUCE-L/R(X): perform the conventional B-REDUCE action.
- SHIFT + U-REDUCE(X) (denoted sh$U(X)$): From the current state p, perform the regular shift action creating new state p_{sh}. And then perform regular unary reduce action from p_{sh} to create new state q whose s_0's constituent label X. In sum, this action returns state q as a newly constructed state.
- B-REDUCE-L/R(X) + U-REDUCE(Y) (denoted bu$_{L/R}(Y)$): From the current state p, perform the regular binary reduce action to create a state pb with s_0's constituent label equaling Y. After that, perform the unary reduce action from p_b to create a new state q with s_0's constituent label X. This action also returns state q as a newly constructed state.
- FINISH (denoted Fin): the conventional FINISH action.

[1] $\Phi(s)$ is the feature function of state s. Based on that, $\Phi_a(j, S)$ is the feature function of current state (j, S) with the action a.

Table 2: A table of baseline feature template

Description	Templates
1-grams	$s_0.t + s_0.c$; $s_0.w + s_0.c$; $s_1.t + s_1.c$; $s_1.w + s_1.c$ $s_2.t + s_2.c$; $s_2.w + s_2.c$; $s_3.t + s_3.c$; $s_3.w + s_3.c$ $q_0.w + q_0.t$; $q_1.w + q_1.c$; $q_2.w + q_2.t$; $q_3.w + q_3.c$ $s_0l.w + s_0l.c$; $s_0u.w + s_0u.c$; $s_0r.w + s_0r.c$ $s_1l.w + s_1l.c$; $s_1u.w + s_1u.c$; $s_1r.w + s_1r.c$
2-grams	$s_0.w + s_1.w$; $s_0.w + s_1.c$; $s_0.c + s_1.w$; $s_0.c + s_1.c$ $s_0.w + q_0.w$; $s_0.w + q_0.t$; $s_0.c + q_0.w$; $s_0.c + q_0.t$ $q_0.w + q_1.w$; $q_0.w + q_1.t$; $q_0.t + q_1.w$; $q_0.t + q_1.t$ $s_1.w + q_0.w$; $s_1.w + q_0.t$; $s_1.c + q_0.w$; $s_1.c + q_0.t$
3-grams	$s_0.c + s_1.c + s_2.c$; $s_0.w + s_1.c + s_2.c$; $s_0.c + s_1.w + s_2.c$; $s_0.c + s_1.c + s_2.w$ $s_0.c + s_1.c + q_0.t$; $s_0.w + s_1.c + q_0.t$; $s_0.c + s_1.w + q_0.t$; $s_0.c + s_1.c + q_0.w$

The deductive system of this modified actions has been shown in figure 2. It is clear that we have removed the unary actions by combining them with shift and binary actions. Therefore, We have only two main types of actions: *shift* or *binary reduce*. This method has two benefits: first, it can guarantee that the number of shift-reduce actions for each possible parse tree will always be $2n$. Second, it does not create any redundant states (such as IDLE) which increase the complexity and sparsity of our model.

2.4 Dynamic programming

(Liang and Kenji, 2010) has proposed a method to apply dynamic programming into shift-reduce dependency parsing by using graph structured-stack of (Masaru, 1991). The key of their method is to merge the equivalent states. In our system, we have adapted this methods into our constituent parsing with some modification. In order to check the equivalence, each state d has been assigned a signature function denoted $sign(d)$, which is similar to kernel features in (Liang and Kenji, 2010). Two states are equivalent if their signature functions return the same values. For each state, the signature function returns the minimum set of atomic values which is enough to construct the features extracted from referred states. More specifically, with the baseline template in table 2, the signature function will return the following set:

- $s_3.c$; $s_2.c$; $s_1.c$; $s_0.c$
- $s_3.start$; $s_2.start$; $s_1.start$; $s_0.start$; $s_0.end$
- $s_3.head$; $s_2.head$; $s_1.head$; $s_0.head$
- $s_1.split$; $s_0.split$
- $s_1l.c$; $s_1r.c$; $s_0l.c$; $s_1r.c$

  Where $start$,end,$head$ are respectively the start,end,head index of referred node. $split$ is the index of the split between left and right child of referred node.

The deductive system of our dynamic programming has been shown as in figure 3 which is modified from the deductive system of (Liang and Kenji, 2010). In dynamic programming, we only need to store the values of signature function instead of the entire stack and queue for each state. There are some fundamental definitions in dynamic shift-reduce parsing:

- Predictor-states $\pi(p)$ are the states which creates p after taking shift actions (both normal and unary shift). It means that every states in $\pi(p)$ can be combined with state p in b-reduce actions as in our deductive system.
- *Shift-states* are the states caused by shift actions (both normal and unary shift).
- *Binary reduce (b-reduce) states* are the states caused by b-reduce actions (both normal and unary b-reduce).

state form	$l : \langle i, j, s_d ... s_0 \rangle : (c, v, \pi)$ $\quad$ l: step; c, v: total and inside costs; π: predictor states
equivalence	$state1 = state2$ $\quad$ iff. $sign(state1) \equiv sign(state2)$
ordering	$state1 < state2$ $\quad$ iff. $(state1.c < state2.c)$ or $(state1.c = state2.c$ and $state1.v < state2.v)$

axiom(p_0)

$$0 : \langle 0, 0, \epsilon \rangle : (0, 0, \emptyset)$$

sh

$$\frac{\overset{state\ r}{l : \langle \_, j, s_d ... s_0 \rangle : (c, \_, \_)}}{l+1 : \langle j, j+1, s_{d-1} ... s_0, w_j \rangle : (c + \xi(r), 0, \{p\})} \quad j < n$$

sh$U(Y)$

$$\frac{\overset{state\ r}{l : \langle \_, j, s_d ... s_0 \rangle : (c, \_, \_)}}{\begin{array}{c} l+1 : \langle j, j+1, s_{d-1} ... s_0,\ \mathrm{Y} \rangle : (c + \xi_u(r), 0, \{p\}) \\ | \\ w_j \end{array}} \quad j < n$$

b$_{L/R}$(X)

$$\frac{\overset{state\ p}{\_ : \langle k, i, s_{d+1} ... s_1 \rangle : (c', v', \pi')} \qquad \overset{state\ q}{l : \langle i, j, s_d ... s_0 \rangle : (c, v, \pi)}}{\begin{array}{c} l+1 : \langle k, j, s_{d+1} ... s_2, \quad \mathrm{X} \quad \rangle : (c' + v + \delta, v' + v + \delta, \pi) \\ \diagup \quad \diagdown \\ s_1 \quad s_0 \end{array}} \quad p \in \pi$$

bu$_{L/R}$(Y)

$$\frac{\overset{state\ p}{\_ : \langle k, i, s_{d+1} ... s_1 \rangle : (c', v', \pi')} \qquad \overset{state\ q}{l : \langle i, j, s_d ... s_0 \rangle : (c, v, \pi)}}{\begin{array}{c} l+1 : \langle k, j, s_{d+1} ... s_2, \quad \mathrm{Y} \quad \rangle : (c' + v + \delta_u, v' + v + \delta_u, \pi) \\ | \\ \mathrm{X} \\ \diagup \quad \diagdown \\ s_1 \quad s_0 \end{array}} \quad p \in \pi$$

goal

$$2n - 1 : \langle 0, n, s_d ... s_0 \rangle : (c, c, \{p_0\})$$

where $\xi = w \cdot \Phi_{sh}$; $\quad \xi_u = w \cdot \Phi_{shU(Y)}$; $\quad \delta = \xi'(p) + \lambda$; $\quad \delta_u = \xi'(p) + \lambda_u$; $\quad \xi'$ = $\xi(p)$ or $\xi_u(p)$;
$\lambda = w \cdot \Phi_{b_{L/R}(X)}(q)$; $\quad \lambda_u = w \cdot \Phi_{bu_{L/R}(X)}(q)$.

Figure 3: The deductive system for our dynamic programming based on (Liang and Kenji, 2010) system. Each state will have: $\langle i, j \rangle$ as the start and end index, and $s_d ... s_0$ as the top-d items in stack.

$$\text{input:} \quad w_0 ... w_{n-1}$$

$$\text{axiom} \quad 0 : \langle 0, \epsilon \rangle : 0$$

$$\text{sh} \quad \frac{l : \langle j, S \rangle : c}{l+1 : \langle j+1, S|w_j \rangle : c + \xi}$$

$$\text{sh}U(Y) \quad \frac{l : \langle j, S \rangle : c}{l+1 : \langle j+1, S|Y(w_j) \rangle : c + \xi_u}$$

$$\text{b}_L(X) \quad \frac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|X(s_1) \rangle : c + \lambda_L}$$

$$\text{b}_R(X) \quad \frac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|X(s_0) \rangle : c + \lambda_R}$$

$$\text{bu}_L(Y) \quad \frac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|Y(s_1) \rangle : c + \lambda_{LU}}$$

$$\text{bu}_L(Y) \quad \frac{l : \langle j, S|s_1|s_0 \rangle : c}{l+1 : \langle j, S|Y(s_0) \rangle : c + \lambda_{RU}}$$

$$Fin \quad \frac{2n-1 : \langle n, s_0 \rangle : c}{2n : \langle n, \emptyset \rangle : c + \rho}$$

where ξ_u, λ_{LU}, λ_{RU} are respectively the shift-unary cost, left binary-unary cost, right binary-unary cost, unary cost.

$$\xi_u = w \cdot \Phi_{shU(Y)}(j, S) \tag{6}$$

$$\lambda_{LU} = w \cdot \Phi_{bu_L(Y)}(j, S) \tag{7}$$

$$\lambda_{RU} = w \cdot \Phi_{bu_R(Y)}(j, S) \tag{8}$$

$$\rho = w \cdot \Phi_{Fin}(2n-1, s_0) \tag{9}$$

Figure 2: Deductive system for our modified shift-reduce constituent parser.

- Total cost c is the real model score of referred state.
- Inside cost v: the Viterbi inside cost of top item (s_0) in referred states.

In dynamic shift-reduce process, if there are two equivalent states, they would be merged together and their predictor states would be united. The merged state will take the total and inside cost of the better state.

3 The influence of feature templates on time complexity

In shift-reduce parsing, it is very important to evaluate the

In shift-reduce parsing, the search space will be very large depending on the feature templates we used. Therefore, in this section, we would like to make some evaluation about the time complexity of feature templates and then design a feature template used for exact search in shift-reduce constituent parsing.

3.1 Evaluation from the baseline feature template

To evaluate the time complexity of shift-reduce parsing system applying this feature template, we will reply on the observation that the baseline feature has focused on top four nodes in stack S (table **??**). With each node, we could consider the parsing process as a CYK process. As we know, dynamic algorithm such as CYK can parse an input sentence within $O(n^3.|G|)$ (cubic time) with unlexicalized case and $O(n^5.|G|)$ with lexicalized case. The table **??** has shown that the baseline feature template uses head word as one of its components, so this is the lexicalized case. Therefore, with the combination of four nodes in the lexicalized case, the parsing system which applies the baseline feature must take a very large time complexity about $O((n^5.|G|)^4) = O(n^{20}.|G|^4)$ in the worst case. In general form, if the feature template focuses on top k nodes in stack S, then we will have the time complexity $O((n^5 * |G|)^k)$ with lexicalized parsing and $O((n^3 * |G|)^k)$ with unlexicalized parsing.

Following the result published in (Kai et al., 2013), the exact search will be possible to perform if we can reduce the time complexity to $O(n^6)$. Comparing this complexity to the general form of time complexity in shift-reduce parsing: $O((n^5 * |G|)^k)$ (lexicalized) and $O((n^3 * |G|)^k)$ (unlexicalized), we will have $k = 1$ (lexicalized) or $k = 2$ (unlexicalized). It means that we have two directions to create our own feature template:

- $k = 1(lexicalized)$: designing a feature template which focuses only on top node in stack S which can use the head word features.
- $k = 2(unlexicalized)$: designing a feature template which focuses on top two nodes in stack S without the head word features.

3.2 Our simplified feature template

As we concluded in the previous section, we will have two strategies of creating a feature template for exact search. With the first approach, the feature template will be so small and lack of information so we created our own feature template based on the second one.

With the second one, the only problem is the lack of head lexical information. Fortunately, there are many studies on unlexicalized constituent parsing which give high parsing accuracy. (Dan and Christopher, 2003b) proposed their first research on unlexicalized parsing. This is a theoretically splitting constituent method which can reach 85.77% F-score. (Slav and Dan, 2007) has proposed out an extension of (Dan and Christopher, 2003b) method which splits the grammar automatically to exploit the latent variables and attain the F-score equaling 90.7%.

As in our system, we have designed a feature template which was inspired by the span features in (David et al., 2014). This is a very simple feature set which relied only on the local features within the span of a node and achieve an amazing performance. In the experiment reported in (David et al., 2014), it has reached an F-score = 89.9% and outperformed Berkeley parser in terms of parsing many different languages. The span features from this article include the following components for a constituent node (table 3):

- The first and last word of the span of node.
- the length of the span of node
- The shape of word sequence within the span of node (illustrated in figure 4):
  - **X** : if the current word's first letter is capital letter.
  - **x** : if the current word is normal word.
  - **N** : if the current word is a number.
  - _ : special character such as [' " , .] is kept.
  - **O** : the other cases.

Our simplified feature template has been shown in table 4. It has focused only on two top nodes in stack S(s_0 and s_1) and did not use the head lexical information. Therefore, based on our previous analysis, the time complexity of this feature template will be $O(n^6 * |G|^2)$. You can see the illustration of comparison between the baseline and simplified feature template in **??**.

Table 3: The sub-components which are used in our simplified feature template

component	explanation
$X.ft$	The first PoS in X's span
$X.fw$	The first word in X's span
$X.lt$	the last PoS in X's span
$X.lw$	the last word in X's span
$X.len$	The length of X's span
$X.shape$	The shape of X's span

4 A* decoder for Shift-Reduce Parsing

A* search is a very well-known algorithm which belongs to the best-first algorithm. The best-first algorithm uses an agenda to store all the processed points in the searching process. In each step, the most promising point will be used to extend until we reach the final point.

However, unlike the regular best first algorithm which uses only the path cost $g(x)$ of each point x in agenda to calculate the best point, A* algorithm also uses an heuristic $h(x)$ which is an admissible heuristic of the path from x to the final point. It means that each point x in A* algorithm will have the score calculated by:

$$score(x) = g(x) + h(x) \tag{10}$$

A* algorithm can guarantee to find the goal with smallest path cost in a shorter time than regular best-first search. To make sure that A* algorithm is efficient, $h(x)$ should satisfy two characters:

- admissible: the heuristic must be greater or equal than the real highest cost we have to take in order to get to the final point: $h(x) \geq h_{real}(x)$
- consistency: $h(x) \leq d(x, y) + h(y)$, for every (x, y) where y is next to x, $d(x, y)$ is the cost from x to y. This is an additional condition.

Therefore, it is plain to see that $h(x)$ is the key of A* algorithm's efficiency.

Table 4: A table of our simplified feature template

Description	Templates
Queue	$q_0.w$+$q_0.t$; $q_1.w$+$q_1.c$; $q_2.w$+$q_2.t$; $q_3.w$+$q_3.c$
s_0's features	$s_0.c$+$s_0.ft$; $s_0.c$+$s_0.fw$; $s_0.c$+$s_0.lt$; $s_0.c$+$s_0.lw$ $s_0.c$+$s_0.ft$+$s_0.lt$; $s_0.c$+$s_0.ft$+$s_0.lw$; $s_0.c$+$s_0.fw$+$s_0.lt$; $s_0.c$+$s_0.fw$+$s_0.lw$ $s_0.c$+$s_0.ft$+$s_0.len$; $s_0.c$+$s_0.fw$+$s_0.len$; $s_0.c$+$s_0.lt$+$s_0.len$; $s_0.c$+$s_0.lw$+$s_0.len$ $s_0.c$+$s_0.ft$+$s_0.lt$+$s_0.len$; $s_0.c$+$s_0.ft$+$s_0.lw$+$s_0.len$ $s_0.c$+$s_0.fw$+$s_0.lt$+$s_0.len$; $s_0.c$+$s_0.fw$+$s_0.lw$+$s_0.len$ $s_0.c$+$s_0.len$; $s_0.c$+$s_0.shape$
s_1's features	$s_1.c$+$s_1.ft$; $s_1.c$+$s_1.fw$; $s_1.c$+$s_1.lt$; $s_1.c$+$s_1.lw$ $s_1.c$+$s_1.ft$+$s_1.lt$; $s_1.c$+$s_1.ft$+$s_1.lw$; $s_1.c$+$s_1.fw$+$s_1.lt$; $s_1.c$+$s_1.fw$+$s_1.lw$ $s_1.c$+$s_1.ft$+$s_1.len$; $s_1.c$+$s_1.fw$+$s_1.len$; $s_1.c$+$s_1.lt$+$s_1.len$; $s_1.c$+$s_1.lw$+$s_1.len$ $s_1.c$+$s_1.ft$+$s_1.lt$+$s_1.len$; $s_1.c$+$s_1.ft$+$s_1.lw$+$s_1.len$ $s_1.c$+$s_1.fw$+$s_1.lt$+$s_1.len$; $s_1.c$+$s_1.fw$+$s_1.lw$+$s_1.len$ $s_1.c$+$s_1.len$; $s_1.c$+$s_1.shape$

4.1 Best first decoder

In order to utilize A* heuristic, we firstly built our best first decoder which is similar the one in (Kai et al., 2013) but for constituent parsing. Let us denote that $BFS(x)$ is the candidate final state returned by the best first decoder. The best first decoder can be described as follows:

- Set up an agenda adding the initial state.
- In each step, popping out the best candidate in agenda and extend it by using the shift-reduce actions. After that, put the newly constructed states back into agenda.
- Looping until the popped candidate from agenda is the final state.

In best first decoder, the model score of each state s is the cost path, the model score of each shift-reduce actions a_i is the transition cost between each state. Let us recall that the model score of state s is the sum of model score of all its shift-reduce actions a_i as in (4). The model score of a_i is calculated as in equation (5). If we use A* decoder, $score(s)$ will be calculated by $g(s) + h(s)$, where $h(s)$ is the heuristic which will be discussed in the next section. The training process of shift-reduce parsing with best first decoder is illustrated in table 5.

$$score(s) = g(s) = \sum_{i=0}^{N} score(a_i) \qquad (11)$$

$$score(a_i) = \Phi(a_i).\vec{w} \qquad (12)$$

The only remain bottle-neck is that the negative score problem happened in the shift-reduce parsing process. In best-first search algorithm, the transition of each shift-reduce actions must be positive. However, due to the learning strategy of averaged perceptron, there still exist a negative value in the weights vector $\vec{w}$, and this may lead to negative score problem. Concerning about this problem, we solved this by adding a certain offset value to the transition score of each actions to keep it positive. Because the number of shift-reduce actions is always $2n$ (section 2.3) in our system, the rank of final states is still the same as before. So it will be able to apply this method.

4.2 Grammar projection

Grammar projection is an interesting idea published in (Dan and Christopher, 2003a) and this may be the first research on A* parsing in our knowledge. In this method, they have projected the original grammar rules to a simpler grammar rules. Each projected production rule always have the PCFG probability that is the maximum value of all the original production rules projecting to it. With this conversion, the outside PCFG score of the parse tree constructed by this projected grammar is always greater or equal to the real outside PCFG score and it can be an heuristic for A* search.

Table 5: Averaged perceptron algorithm with best first search decoder.

Inputs	Training examples (x_i, y_i)
Initialization	$\vec{w}$ = 0
Output	averaged weights $\vec{w}$
Algorithm	For $t = 1...T, i = 1...n$ — Calculate $F(x_i) = BFS(x_i)$ — If $(F(x_i) \neq y_i)$ then $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$

Table 6: The experiment on the development set.

System	F-score on BF	parsing speed on BF	F-score on SF	parsing speed on SF
beam search decoder with $beam = 16$	89.07%	25 sentences/s	88.72%	30 sentences/s
beam search decoder with $beam = 32$	89.87%	10 sentences/s	89.33%	13 sentences/s
beam search decoder with $beam = 64$	90.3%	3.4 sentences/s	90.15%	4.5 sentences/s
beam search decoder with $beam = 128$	90.3%	1.05 sentences/s	90.23%	1.36 sentences/s
$A*$ decoder	N/A	N/A	90.7%	1.2 sentences/s

In our system, we design our own grammar projection based on the filter projection from (Dan and Christopher, 2003a) with some modification which is suitable with shift-reduce parsing:

- project all the complete constituent tags such as [NP, VP, ADJP...] into one single label called [XTAG].
- keeping all the incomplete constituent tags such as [NP*, AP*, ADJP*] and the PoS tag as before.

Corresponding to this projection, the features will also be projected in our system. The projected features will have the weight equaling the maximum value of all features which project it. For example, there are shift features whose types are $s_0.fw$+$s_0.c$ with values: ([google+VP]; [google+NP]; [google+AP]) with the weights (14.65; 5.67; 8.2), respectively. They will be projected into one value: [google+XTAG] with the weight equaling to 14.65.

In our A* decoder, with each state s, we will perform the best first search with these projected grammar rules and features to find the heuristic final state f'. After projecting, the time complexity of decoding proces will be reduced significantly and can be performed by best first search. The transition score from s to f' will be the heuristic of s in A* search. Because the projected features is the maximum summarization of the original features in terms of weight values, the heuristic transition score will always be greater or equal than the real transition score. It means that the grammar projection heuristic presented here is an *admissible* heuristic.

4.3 The less constituent heuristic

The *less constituent* is an heuristic which have been proposed by us based on this observation: if we reduce the number of nodes focused in the feature template, then we can significantly reduce the time com-

plexity. Therefore, if we focused on only s_0 instead of both s_0 and s_1, the time complexity for calculation will be reduced to $O(n^3.|G|)$ which can be decoded by best first search (similar to normal CYK algorithm). As in grammar projection, the heuristic transition score is calculated by best first search after considering less constituent will be the heuristic for A* search. More specifically, to calculate this heuristic, the s_0's features in table 4 will still be the same but the s_1's features will be ignored and carry the maximum weight value of their types.

For example, the current state s have three attached features: [$s_0.c$+$s_0.ft$], [$s_0.c$+$s_0.fw$] and [$s_1.c$+$s_1.fw$]. Then the two features whose type is [$s_0.c$+$s_0.ft$], [$s_0.c$+$s_0.fw$] will be extracted from s and calculated normally, but the [$s_1.c$+$s_1.fw$] feature of s will be assigned the maximum weight value of all features whose type is [$s_1.c$+$s_1.fw$]. Because the s_1's weights after being ignored are always greater or equal (maximum) than the original s_1's weights, it is clear to see that the heuristic transition score calculated by this way is always greater or equal than the real transition score and this heuristic is also admissible.

4.4 Our cascaded heuristic

Both of those two above heuristics have their own problem: calculating the *grammar projection* heuristic will have time complexity = $O(n^6 * |g|^2)$ where g is the projected grammar, and calculate the *less constituent* heuristic will have time complexity = $O(n^3 * |G|)$. Both of calculation may be a little expensive so we combine them together to reduce the total time complexity.

The process of calculating our heuristic could be described as follows: we use the *less constituent* method to calculate A* heuristic for the original best first decoder. And in the process of calculating less constituent heuristic, we use the *grammar projection* to guide the best first search go faster. As you can see, this heuristic is a cascaded type in which an A* heuristic is calculated by another A* heuristic. This method will lead to significant speed-up: the time complexity of calculating *less constituent* will be as fast as the *filter* heuristic in (Dan and Christopher, 2003a), and the *less constituent* is also an effective heuristic which will lead the original best first decoder go faster to the best final state.

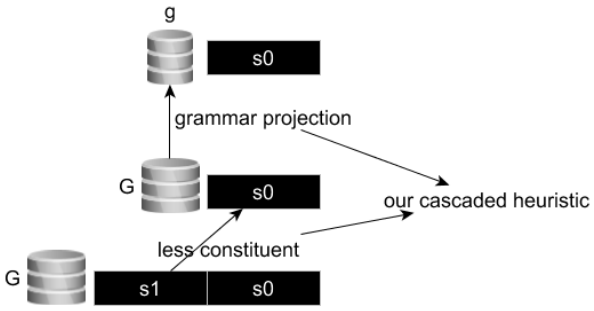

Figure 4: The span feature from adapted from (David et al., 2014).

5 Experiment

5.1 Preparation

We evaluate our A* shift-reduce parser on the Wall Street Journal (WSJ) corpus of the Penn Treebank project with the standard split: sections 2-21 were used for training, section 22 as a development set, and section 23 was used for testing. All our experiments reported in this paper was performed on [our computer configuration].

We used the head finder of (Michael, 1999) to define the head constituent tag for each phrase in the corpus, and adapted the binarization of (Yue and Stephen, 2009) to binarize our grammar rules. We also used Stanford PoS tagger (with the accuracy = 97%) to produce the tagged sentence as an input for our shift-reduce constituent parsing. To evaluate the ParseVal F-score, we utilize EVALB program.

5.2 The experiment results on development set

On the development set, we make an experiment of comparing our proposal A* decoder with beam search decoder. We also test these decoders in both baseline and simplified feature template. The experiment results can be viewed in table 6.

5.3 The experiment results on test set

On the test set, we make an experiment for comparing our A* parser with the state-of-the-art parsing systems. The results of this experiment are shown in table 7.

Table 7: Results of the final experiment on test set to compare our A* shift reduce parser with other parsing system.

System	F-score
Johnson and Charniak (2005)	91.4%
Mc Closky (2006)	92.1%
Collins (1999), Bikel (2004)	87.7%
Berkeley Parser	90.1%
Stanford Parser	90.4%
SSN parser - Henderson(2004)	89.4%
Zhang Yue - baseline (2012)	89.8%
This paper	91.1%

6 Conclusion and future works

References

Dan, K. and Christopher, D. (2003a). A* parsing: Fast exact viterbi parse selection. In *Proceedings of HLT-NAACL*.

Dan, K. and Christopher, D. (2003b). Accurate unlexicalized parsing. In *Proceedings of ACL*.

David, H., Greg, D., and Dan, K. (2014). Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 228–337, Baltimore, Maryland.

Kai, Z., Jame, C., and Liang, H. (2013). Optimal incremental parsing via best-first dynamic programming. In *Proceedings of EMNLP 2013*.

Kenji, S. and Alon, L. (2006). A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL*, pages 691–698.

Liang, H. and Kenji, S. (2010). Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL 2010*.

Masaru, T. (1991). Generalized lr parsing. Kluwer Academic.

Mengqiu, Wang, K. S. and Teruko, M. (2006). A fast, accurate deterministic parser for chinese. In *Proceedings of COLING/ACL*, page 425432, Sydney, Australia.

Michael, C. (1999). Head-driven statistical models for natural language parsing. In *Ph.D. thesis*, University of Pennsylvania.

Michael, C. and Brian, R. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, Stroudsburg, PA, USA.

Slav, P. and Dan, K. (2007). Improved inference for unlexicalized parsing. In *Proceedings of HLT/NAACL*, pages 404–411, Rochester, New York.

Yue, Z. and Stephen, C. (2009). Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of IWPT*, Paris, France.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., and Zhu, J. (2012). Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria. Association for Computational Linguistics.