An AMR Aligner Tuned by Transition-based Parser

Yijia Liu, Wanxiang Che[*], Bo Zheng, Bing Qin, Ting Liu
Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, China
{yjliu,car,bzheng,qinb,tliu}@ir.hit.edu.cn

Abstract

In this paper, we propose a new rich resource enhanced AMR aligner which produces multiple alignments and a new transition system for AMR parsing along with its oracle parser. Our aligner is further tuned by our oracle parser via picking the alignment that leads to the highest-scored achievable AMR graph. Experimental results show that our aligner outperforms the rule-based aligner in previous work by achieving higher alignment F1 score and consistently improving two open-sourced AMR parsers. Based on our aligner and transition system, we develop a transition-based AMR parser that parses a sentence into its AMR graph directly. An ensemble of our parsers with only words and POS tags as input leads to 68.4 Smatch F1 score, which outperforms the parser of Wang and Xue (2017).

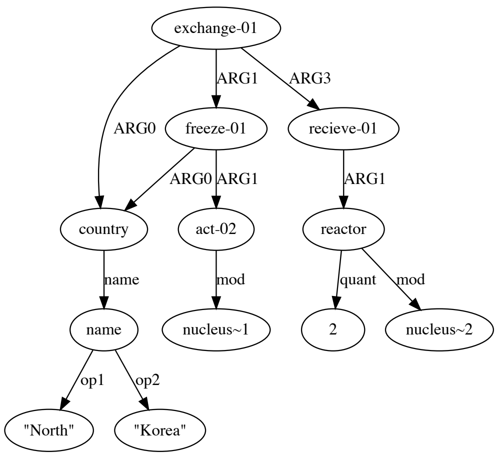

Figure 1: AMR graph for the sentence *"North Korea froze its nuclear actions in exchange for two nuclear reactors."*

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic representation which encodes the meaning of a sentence in a rooted and directed graph, whose nodes are abstract semantic concepts and edges are semantic relations between concepts (see Figure 1 for an example). Parsing a sentence into its AMR graph has drawn a lot of research attention in recent years with a number of parsers being developed (Flanigan et al., 2014; Wang et al., 2015b; Pust et al., 2015; Artzi et al., 2015; Peng et al., 2015; Zhou et al., 2016; Goodman et al., 2016; Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017; Foland and Martin, 2017; Konstas et al., 2017).

The nature of abstracting away the association between a concept and a span of words complicates the training of the AMR parser. A word-concept aligner is required to derive such association from the sentence-AMR-graph pair and the alignment output is then used as reference to train the AMR parser. In previous works, such alignment is extracted by either greedily applying a set of heuristic rules (Flanigan et al., 2014) or adopting the unsupervised word alignment technique from machine translation (Pourdamghani et al., 2014; Wang and Xue, 2017).

The rule-based aligner – JAMR aligner proposed by Flanigan et al. (2014) is widely used in previous works thanks to its flexibility of incorporating additional linguistic resources like WordNet. However, achieving good alignments with the JAMR aligner still faces some difficult challenges. The first challenge is deriving an optimal alignment in ambiguous situations. Taking the sentence-AMR-graph pair in Figure 1 for example, the JAMR aligner doesn't distinguish between the two "*nuclear*"s in the sentence and can yield sub-optimal alignment in which the first "*nuclear*" is aligned to the `nucleus~2` concept. The second challenge is recalling more semantically matched word-concept pair without harming the

[*] Email corresponding.

alignment precision. The JAMR aligner adopts a rule that aligns the word-concept pair which at least have a common longest prefix of 4 characters, but omitting the shorter cases like aligning the word "*actions*" to the concept `act-01` and the semantically matched cases like aligning the word "*example*" to the concept `exemplify-01`. The final challenge which is faced by both the rule-based and unsupervised aligners is tuning the alignment with downstream parser learning. Previous works treated the alignment as a fixed input. Its quality is never evaluated and its alternatives are never explored. All these challenges make the JAMR aligner achieve only an alignment F1 score of about 90% and influence the performance of the trained AMR parsers.

In this paper, we propose a novel method to solve these challenges and improve the word-to-concept alignment, which further improves the AMR parsing performance. A rule-based aligner and a transition-based oracle AMR parser lie in the core of our method. For the aligner part, we incorporate rich semantic resources into the JAMR aligner to recall more word-concept pairs and cancel its greedily aligning process. This leads to multiple alignment outputs with higher recall but lower precision. For the parser part, we propose a new transition system that can parse the raw sentence into AMR graph directly. Meanwhile, a new oracle algorithm is proposed which produces the best achievable AMR graph from an alignment. Our aligner is tuned by our oracle parser by feeding the alignments to the oracle parser and picking the one which leads to the highest Smatch F1 score (Cai and Knight, 2013). The chosen alignment is used in downstream training of the AMR parser. Based on the newly proposed aligner and transition system, we develop a transition-based parser that directly parses a sentence into its AMR graph and it can be easily improved through ensemble thanks to its simplicity.

We conduct experiments on LDC2014T12 dataset.[1] Both intrinsic and extrinsic evaluations are performed on our aligner. In the intrinsic evaluation, our aligner achieves an alignment F1 score of 95.2%. In the extrinsic evaluation, we replace the JAMR aligner with ours in two open-sourced AMR parsers, which leads to consistent improvements on both parsers. We also evaluate our transition-based parser on the same dataset. Using both our aligner and ensemble, a score of 68.1 Smatch F1 is achieved without any additional resources, which is comparable to the parser of Wang and Xue (2017). With additional part-of-speech (POS) tags, our ensemble parser achieves 68.4 Smatch F1 score and outperforms that of Wang and Xue (2017).

The contributions of this paper come in two folds:

- We propose a new AMR aligner (§3) which recalls more semantically matched pairs and produces multiple alignments. We also propose a new transition system for AMR parsing (§4.1) and use its oracle (§4.2) to pick the alignment that leads to the highest-scored achievable AMR graph (§4.3). Both intrinsic and extrinsic evaluations (§5) show the effectiveness of our aligner by achieving higher F1 score and consistently improving two open-sourced AMR parsers.

- We build a new transition-based parser (§4.4) upon our aligner and transition system which directly parses a raw sentence into its AMR graph. Through simple ensemble, our parser achieves 68.4 Smatch F1 score with only words and POS tags as input (§6) and outperforms the parser of Wang and Xue (2017).

Our code and the alignments for LDC2014T12 dataset are publicly available at `https://github.com/Oneplus/tamr`

2 Related Work

AMR Parsers. AMR parsing maps a natural language sentence into its AMR graph. Most current parsers construct the AMR graph in a two-staged manner which first identifies concepts (nodes in the graph) from the input sentence, then identifies relations (edges in the graph) between the identified concepts. Flanigan et al. (2014) and their follow-up works (Flanigan et al., 2016; Zhou et al., 2016) model the parsing problem as finding the maximum spanning connected graph. Wang et al. (2015b) proposes to greedily transduce the dependency tree into AMR graph and a bunch of works (Wang et al., 2015a; Goodman et al., 2016; Wang and Xue, 2017) further improve the transducer's performance with rich features and imitation learning.[2] Transition-based methods

[1] `catalog.ldc.upenn.edu/ldc2014t12`

[2] Wang et al. (2015b) and the follow-up works refer their transducing process as "transition-based". However, to dis-

that directly parse an input sentence into its AMR graph have also been studied (Ballesteros and Al-Onaizan, 2017; Damonte et al., 2017). In these works, the concept identification and relation identification are performed jointly.

An aligner which maps a span of words into its concept serves to the generation of training data for the concept identifier, thus is important to the parser training. Missing or incorrect alignments lead to poor concept identification, which then hurt the overall AMR parsing performance. Besides the typical two-staged methods, the aligner also works in some other AMR parsing algorithms like that using syntax-based machine translation (Pust et al., 2015), sequence-to-sequence (Peng et al., 2017; Konstas et al., 2017), Hyperedge Replacement Grammar (Peng et al., 2015) and Combinatory Category Grammar (Artzi et al., 2015).

Previous aligner works solve the alignment problem in two different ways. The rule-based aligner (Flanigan et al., 2014) defines a set of heuristic rules which align a span of words to the graph fragment and greedily applies these rules. The unsupervised aligner (Pourdamghani et al., 2014; Wang and Xue, 2017) uncovers the word-to-concept alignment from the linearized AMR graph through EM. All these approaches yield a single alignment for one sentence and its effect on the downstream parsing is not considered.

JAMR Aligner (Flanigan et al., 2014). Two components exist in the JAMR aligner: 1) a set of heuristic rules and 2) a greedy search process.

The heuristic rules in the JAMR aligner are a set of indicator functions $\rho(c, w_{s,e})$ which take a concept c and a span of words $w_{s,e}$ starting from s and ending with e as input and return whether they should be aligned. These rules can be categorized into *matching rules* and *updating rules*. The matching rules directly compare c with $w_{s,e}$ and determine if they should be aligned. The updating rules first retrieve the concept c' that $w_{s,e}$ aligns, then determine if c and $w_{s,e}$ should be aligned by checking whether c and c' meet some conditions. Here, we illustrate how *update rules* work by applying a rule named **Entity Type** on the AMR graph in Figure 1 as an example. When determining if the entity type concept `country` should be aligned to "*North Korea*", the **Entity Type** rule first retrieve that this span is aligned to the fragment `(name :op1 "North" :op2 "Korea")`, then determine if they are aligned by checking if `name` is the tail concept of `country`.

The greedy search process applies rules in a manually defined order. The results are mutually exclusive which means once a graph fragment is aligned by one rule, it cannot be realigned. By doing so, conflicts between the alignments produced by different rules are resolved. Flanigan et al. (2014) didn't talk about the principle of orders but it generally follows the principle that 1) the matching rules have higher priorities than the updating rules, and 2) exact matching rules have higher priorities than the fuzzy matching rules.

3 Enhanced Rule-based Aligner

3.1 Enhancing Aligner with Rich Semantic Resources

Error propagates in the greedy search process. An alignment error can lead to future errors because of the dependencies and mutual exclusions between rules. In the JAMR aligner, rules that recall more alignments but introduce errors are carefully opted out and it influences the aligner's performance. Our motivation is to use rich semantic resources to recall more alignments. Instead of resolving the resulted conflicts and errors by greedy search, we keep the multiple alignments produced by the aligner and let a parser decide the best alignment.

In this paper, we use two kinds of semantic resources to recall more alignments, which include the similarity drawn from *Glove embedding* (Pennington et al., 2014)[3] and the *morphosemantic* database (Fellbaum et al., 2009) in the WordNet project[4]. Two additional matching schemes **semantic match** and **morphological match** are proposed as:

Semantic Match. *Glove embedding* encodes a word into its vector representation. We define *semantic match* of a concept as a word in the sentence that has a cosine similarity greater than 0.7 in the embedding space with the concept striping off trailing number (e.g. `run-01` $\rightarrow$ `run`).

Morphological Match. *Morphosemantic* is a database that contains links among derivational

tinguish their work with that of Damonte et al. (2017) and Ballesteros and Al-Onaizan (2017), we use the term "transduce" instead.

[3] nlp.stanford.edu/projects/glove/

[4] wordnet.princeton.edu/wordnet/download/standoff/

(Semantic Named Entity) Applies to `name` concepts and their `opn` children. Matches a span that matches the semantic match of each child in numerical order.

(Morphological Named Entity) Applies to `name` concepts and their `opn` children. Matches a span that matches the morphological match of each child in numerical order.

(Semantic Concept) Applies to any concept. Strips off trailing '-[0-9]+' from the concept, and matches any semantic matching word.

(Morphological Concept) Applies to any concept. Strips off trailing '-[0-9]+' from the concept, and matches any morphological matching word or WordNet lemma.

Table 1: The extended rules.

links connecting noun and verb senses (e.g., "*example*" and `exemplify`). We define morphological match of a concept as a word in the sentence having the (*word*, `concept`) link in the database.

By defining the **semantic match** and **morphological match**, we extend the rules in Flanigan et al. (2014) with four additional matching rules as shown in Table 1. These rules are intended to recall the concepts or entities which either semantically resemble a span of words but differ in the surface form, or match a span of words in their morphological derivation.

3.2 Producing Multiple Alignments

Using the rules in the JAMR aligner along with our four extended matching rules, we propose an algorithm to draw multiple alignments from a pair of sentence and AMR graph and it is shown in Algorithm 1. In this algorithm, A_c denotes the set of candidate alignments for a graph fragment c, in which each alignment is represented as a tuple (s, e, c') where s denotes the starting position, e denotes the ending position, and c' denotes the concept that lead to this alignment. At the beginning, A_c is initialized as an empty set (line 1 to 2). Then all the matching rules are tried to align a span of words to that fragment (line 3 to 7). After applying all the matching rules, all the updating rules are repeatedly applied until no new alignment is generated in one iteration (line 8 to 16). During applying the updating rules, we keep track of the dependencies between fragments. Finally, all the possible combination of the alignments are enumerated without considering the one that violates the fragment dependencies (line 17 to 26).

4 Transition-based AMR Parser

Our enhanced rule-based aligner produces multiple alignments, and we would like to use our

Algorithm 1: Our alignment algorithm.

```
Input: An AMR graph with a set of graph fragments $C$;
       a sentence $W$; a set of matching rules $\mathrm{P}_M$; and
       a set of updating rules $\mathrm{P}_U$.
Output: a set of alignments $\mathcal{A}$.
1  for $c \in C$ do
2      $A_c \leftarrow \emptyset$;
3  for $\rho_M \in \mathrm{P}_M$ do
4      for $w_{s,e} \leftarrow$ spans($W$) do
5          for $c \in C$ do
6              if $\rho_M(c, w_{s,e})$ then
7                  $A_c \leftarrow A_c \cup (s, e, \text{nil})$;
8  updated $\leftarrow$ true ;
9  while updated is true do
10     updated $\leftarrow$ false;
11     for $\rho_U \in \mathrm{P}_U$ do
12         for $c, c' \in C \times C$ do
13             for $(s, e, d) \in A'_c$ do
14                 if $\rho_U(c, w_{s,e}) \wedge (s, e, c') \notin A_c$ then
15                     $A_c \leftarrow A_c \cup (s, e, c')$;
16                     updated $\leftarrow$ true;
17 $\mathcal{A} \leftarrow \emptyset$ ;
18 for $(a_1, ..., a_c) \in$ CartesianProduct$(A_1, ..., A_{|C|})$ do
19     legal $\leftarrow$ true;
20     for $a \in (a_1, ..., a_c)$ do
21         $(s, e, c') \leftarrow a$;
22         $(s', e', d) \leftarrow a_{c'}$;
23         if $s \neq s' \wedge e \neq e'$ then
24             legal $\leftarrow$ false ;
25     if legal then
26         $\mathcal{A} \leftarrow \mathcal{A} \cup (a_1, ..., a_c)$;
```

parser to evaluate their qualities. A parameterized parser does not accomplish such goal because training its parameters depends on the aligner's outputs. A deterministic parser works in this situation but is required to consider the association between concepts and spans. This stops the deterministic parsers which build AMR graph only from the derived concepts[5] from being used because they do not distinguish alignments that yields to the same set of concepts.[6]

This discussion shows that to evaluate the quality of an alignment, we need a deterministic (oracle) parser which builds the AMR graph from the raw sentence. Ballesteros and Al-Onaizan (2017) presented a transition-based parser that directly parses a sentence into its AMR graph. A transition system which extends the swap-based dependency parsing system to handle AMR non-projectivities (Damonte et al., 2017) was proposed in their work.

[5]e.g. the reference relation identifier in Flanigan et al. (2014) and the oracle transducer in Wang et al. (2015b).

[6]recall the "*nuclear*" example in Section 1.

Transition	Current State	Resulting State	Description
DROP	$[\sigma \mid \texttt{s}_0,\ \delta,\ b_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \beta,\ A]$	pops out the word that doesn't convey any semantics (e.g., function words and punctuations).
MERGE	$[\sigma \mid \texttt{s}_0,\ \delta,\ b_0 \mid b_1 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ b_0\_b_1 \mid \beta,\ A]$	concatenates a sequence of words into a span, which can be derived as a named entity (`name`) or `date-entity`.
CONFIRM(`c`)	$[\sigma \mid \texttt{s}_0,\ \delta,\ b_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{c} \mid \beta,\ A]$	derives the first element of the buffer (a word or span) into a concept `c`.
ENTITY(`c`)	$[\sigma \mid \texttt{s}_0,\ \delta,\ b_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{c} \mid \beta,\ A \cup \text{relations}(c)]$	a special form of CONFIRM that derives the first element into an entity and builds the internal entity AMR fragment.
NEW(`c`)	$[\sigma \mid \texttt{s}_0,\ \delta,\ \text{b}_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{c} \mid \texttt{b}_0 \mid \beta,\ A]$	generates a new concept `c` and pushes it to the front of the buffer.
LEFT(`r`)	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{b}_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{b}_0 \mid \beta,\ A \cup \{\texttt{s}_0 \xleftarrow{\texttt{r}} \texttt{b}_0\}]$	links a relation `r` between the top concepts on the stack and the buffer.
RIGHT(`r`)	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{b}_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{b}_0 \mid \beta,\ A \cup \{\texttt{s}_0 \xrightarrow{\texttt{r}} \texttt{b}_0\}]$	
CACHE	$[\sigma \mid \texttt{s}_0,\ \delta,\ \text{b}_0 \mid \beta,\ A]$	$[\sigma,\ \texttt{s}_0 \mid \delta,\ \text{b}_0 \mid \beta,\ A]$	passes the top concept of the stack onto the deque.
SHIFT	$[\sigma \mid \texttt{s}_0,\ \delta,\ \texttt{b}_0 \mid \beta,\ A]$	$[\sigma \mid \texttt{s}_0 \mid \delta \mid \texttt{b}_0,\ [\,],\ \beta,\ A]$	shifts the first concept of the buffer onto the stack along with those on the deque.
REDUCE	$[\sigma \mid \texttt{s}_0,\ \delta,\ \text{b}_0 \mid \beta,\ A]$	$[\sigma,\ \delta,\ \text{b}_0 \mid \beta,\ A]$	pops the top concept of the stack.

Table 2: The transition system. The letters in `monospace` font represent the concepts, the *italic* letters represent the word, and the letters in normal font are either concepts or words.

Their work presented the possibility for the oracle parser, but their oracle parser was not touched explicitly. What's more, in the non-projective dependency parsing, Choi and McCallum (2013)'s extension to the list-based system (Nivre, 2008) with caching mechanism achieves expected linear time complexity and requires fewer actions to parse a non-projective tree than the swap-based system. Their extension to transition-based AMR parsing is worth studying.

In this paper, we propose to extend Choi and McCallum (2013)'s transition system to AMR parsing and present the corresponding oracle parser. The oracle parser is used for tuning our aligner and training our parser. We also present a comprehensive comparison of our system with that of Ballesteros and Al-Onaizan (2017) in Section 6.3.

4.1 List-based Extension for AMR Parsing

We follow Choi and McCallum (2013) and define a state in our transition system as a quadruple $s = (\sigma, \delta, \beta, A)$, where σ is a stack holding processed words, δ is a deque holding words popped out of σ that will be pushed back in the future, and β is a buffer holding unprocessed words. A is a set of labeled relations. A set of actions is defined to parse sentence into AMR graph. Table 2 gives a formal illustration of these actions and how they work. The first five actions in Table 2 are our extended actions, and they are used to deriving concepts from the input sentence.

4.2 Oracle Parser

Given an alignment and the gold standard AMR graph, we can build the best AMR graph by repeatedly applying one of these actions and this is what we called *oracle* parser. Before running the oracle parser, we first remove the concepts which aren't aligned with any span of words from the AMR graph. During running the oracle parser, for a state $s = (\sigma \mid \texttt{s}_0,\ \delta,\ \text{b}_0 \mid \text{b}_1 \mid \beta,\ A)$, our oracle parser decides which action to apply by checking the following conditions one by one.

1. If b_0 is a word and it doesn't align to any concept, perform DROP.
2. If b_1 is within a span in the alignment, perform MERGE.
3. If b_0 is a word or span and it only aligns to one entity concept `c`, perform ENTITY(`c`).
4. If b_0 is a word or span and it aligns to one or more concepts, perform CONFIRM(`c`) where `c` is the concept b_0 aligns and has the longest graph distance to the root.
5. If b_0 is a concept and its head concept c has the same alignment as b_0, perform NEW(`c`).
6. If b_0 is a concept and there is an unprocessed edge `r` between $\texttt{s}_0$ and $\texttt{t}_0$, perform LEFT(`r`) or RIGHT(`r`) according to `r`'s direction.

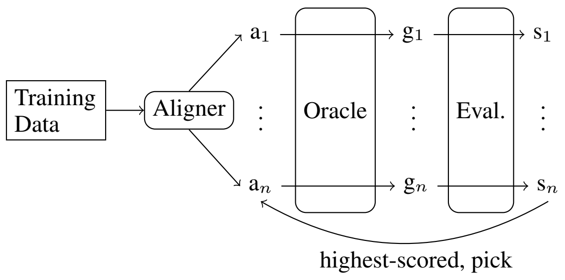

Figure 2: The workflow of tuning the aligner with the oracle parser. a_i denotes the i-th alignment, g_i denotes the i-th AMR graph, and s_i denotes the score of the i-th AMR graph.

7. If s_0 has unprocessed edge, perform CACHE.
8. If s_0 doesn't have unprocessed edge, perform REDUCE.
9. perform SHIFT.

We test our oracle parser on the *hand-align* data created by Flanigan et al. (2014) and it achieves 97.4 Smatch F1 score.[7] Besides the errors resulted from incorrect manual alignments, entity errors made by the limitation of our ENTITY(c) action count a lot. Since our ENTITY action directly converts the surface form of a word span into an entity. It cannot correctly generate entity names when they require derivation,[8] or where tokenization errors exist.[9]

4.3 Tune the Aligner with Oracle Parser

Using our oracle parser, we tune the aligner by picking the alignment which leads to the highest-scored AMR graph from the set of candidates (see Figure 2 for the workflow). When more than one alignment achieve the highest score, we choose the one with the smallest number of actions. Intuitively, choosing the one with the smallest number of actions will encourage structurally coherent alignment[10] because coherent alignment requires fewer CACHE actions.

4.4 Parsing Model

Based on our aligner and transition system, we propose a transition-based parser which parse the raw sentence directly into its AMR graph. In this paper, we follow Ballesteros and Al-Onaizan (2017) and use StackLSTM (Dyer et al., 2015) to model the states. The score of a transition action a on state s is calculated as

$$p(a|s) = \frac{\exp\{g_a \cdot \textsc{StackLSTM}(s) + b_a\}}{\sum_{a'} \exp\{g_{a'} \cdot \textsc{StackLSTM}(s) + b_{a'}\}},$$

where $\textsc{StackLSTM}(s)$ encodes the state s into a vector and g_a is the embedding vector of action a. We encourage the reader to refer Ballesteros and Al-Onaizan (2017) for more details.

Ensemble. Ensemble has been shown as an effective way of improving the neural model's performance (He et al., 2017). Since the transition-based parser directly parse a sentence into its AMR graph, ensemble of several parsers is easier compared to the two-staged AMR parsers. In this paper, we ensemble the parsers trained with different initialization by averaging their probability distribution over the actions.

5 Alignment Experiments

5.1 Settings

We evaluate our aligner on the LDC2014T12 dataset. Two kinds of evaluations are carried out including the *intrinsic* and *extrinsic* evaluations.

For the *intrinsic* evaluation, we follow Flanigan et al. (2014) and evaluate the F1 score of the alignments produced by our aligner against the manually aligned data created in their work (*hand-align*). We also use our oracle parser's performance as an *intrinsic* evaluation assuming that better alignment leads to higher scored oracle parser.

For the *extrinsic* evaluation, we plug our alignment into two open-sourced AMR parsers: 1) JAMR (Flanigan et al., 2014, 2016) and 2) CAMR (Wang et al., 2015b,a) and evaluate the final performances of the AMR parsers on both the newswire proportion and the entire dataset of LDC2014T12. We use the configuration in Flanigan et al. (2016) for JAMR and the configuration in Wang et al. (2015a) without semantic role labeling (SRL) features for CAMR.

5.2 Results

Intrinsic Evaluation. Table 3 shows the intrinsic evaluation results, in which our alignment intrinsically outperforms JAMR aligner by achieving better alignment F1 score and leading to a higher scored oracle parser.

[7] Since some alignments in *hand-align* were created on incorrect AMR annotations, we filter out them and only use the correct subset which has 136 pairs of alignment and AMR graph. This data is also used in our intrinsic evaluation.

[8] e.g., "*North Koreans*" cannot be parsed into `(name :op1 "North" :op2 "Korea")`

[9] e.g., "*Wi Sung - lac*" cannot be parsed into `(name :op1 "Wi" :op2 "Sung-lac")`

[10] e.g. the first "*nuclear*" aligned to `nucleus~1` in Fig. 1

Aligner	Alignment F1 (on hand-align)	Oracle's Smatch (on dev. dataset)
JAMR	90.6	91.7
Our	95.2	94.7

Table 3: The intrinsic evaluation results.

model	newswire	all
JAMR parser: Word, POS, NER, DEP		
+ JAMR aligner	71.3	65.9
+ Our aligner	73.1	67.6
CAMR parser: Word, POS, NER, DEP		
+ JAMR aligner	68.4	64.6
+ Our aligner	68.8	65.1

Table 4: The parsing results.

Extrinsic Evaluation. Table 4 shows the results. From this table, we can see that our alignment consistently improves all the parsers by a margin ranging from 0.5 to 1.7. Both the intrinsic and the extrinsic evaluations show the effectiveness our aligner.

5.3 Ablation

To have a better understanding of our aligner, we conduct ablation test by removing the *semantic matching* and *oracle parser tuning* respectively and retrain the JAMR parser on the newswire proportion. The results are shown in Table 5. From this table, we can see that removing either of these components harms the performance. Removing *oracle parser tuning* leads to severe performance drop and the score is even lower than that with JAMR aligner. We address this observation to that alignment noise is introduced by the semantic matching especially by the word embedding similarity component. Without filtering the noise by our oracle parser, just introducing more matching rules will harm the performance.

6 Parsing Experiments

6.1 Settings

We use the same settings in our aligner extrinsic evaluation for the experiments on our transition-based parser. For the input to the parser, we tried two settings: 1) using only words as input, and 2) using words and POS tags as input. Automatic POS tags are assigned with Stanford POS tagger (Manning et al., 2014). Word embedding from Ling et al. (2015) is used in the same way with Ballesteros and Al-Onaizan (2017). To opt out the effect of different initialization in training the neural network, we run 10 differently seeded runs and report their average performance following Reimers and Gurevych (2017).

model	newswire
JAMR parser + Our aligner	73.1
- Semantic matching	72.7
- Oracle Parser Tuning	67.6
JAMR parser + JAMR aligner	71.3

Table 5: The ablation test results.

6.2 Results

Table 6 shows the performance of our transition-based parser along with comparison to the parsers in the previous works. When compared with our transition-based counterpart (Ballesteros and Al-Onaizan, 2017), our word-only model outperforms theirs using the same JAMR alignment. The same trend is witnessed using words and POS tags as input. When replacing the JAMR alignments with ours, the parsing performances are improved in the same way as in Table 4, which further confirms the effectiveness of our aligner.

The second block in Table 6 shows the results of our ensemble parser, in which ensemble significantly improves the performance and more parsers ensembled, more improvements are achieved. An ensemble of 10 parsers with only words as input achieves 68.1 Smatch F1 score which is comparable to the AMR parser of Wang and Xue (2017). Using the minimal amount of additional syntactic information – POS tags, the performance of the ensemble of 10 parsers is further pushed to 68.4, which surpasses that of Wang and Xue (2017) which relied on named entity recognition (NER) and dependency parsing (DEP).

A further study on the speed shows that our 10 parser ensemble can parse 43 tokens per second which is faster than JAMR (7 tokens/sec.) and CAMR (24 tokens/sec.) thanks to the simplicity of our model and independence of preprocessing, like NER and DEP.[11]

6.3 Comparison to Ballesteros and Al-Onaizan (2017)

To explain the improved performance against Ballesteros and Al-Onaizan (2017) in Table 6, we

[11]In our speed comparison, we also count the time of preprocessing for JAMR and CAMR. All the comparison is performed in the same single-threaded settings.

model	newswire	all
Our single parser: Word only		
+ JAMR aligner	68.6	63.9
+ Our aligner	69.3	64.7
Our single parser: Word, POS		
+ JAMR aligner	68.8	64.6
+ Our aligner	69.8	65.2
Our ensemble: Word only + Our aligner		
x3	71.9	67.4
x10	72.5	68.1
Our ensemble: Word, POS + Our aligner		
x3	72.5	67.7
x10	73.3	**68.4**
BA17: Word only†	68	63
+ POS	68	63
+ POS, DEP	69	64
Damonte et al. (2017)‡	-	66
Artzi et al. (2015)	66.3	-
Wang et al. (2015a)	70	66
Pust et al. (2015)	-	67.1
Zhou et al. (2016)	71	66
Goodman et al. (2016)	70	-
Wang and Xue (2017)	-	68.1

Table 6: The parsing results. xn denotes the ensemble of n differently initialized parsers. The difference in rounding is due to previous works report differently rounded results. † BA17 represents the result of Ballesteros and Al-Onaizan (2017), ‡ Damonte et al. (2017)'s result is drawn from Ballesteros and Al-Onaizan (2017).

give a comprehensive comparison between our transition system and that of Ballesteros and Al-Onaizan (2017).

Capability. In both these two systems, a span of words can only be derived into concept for one time. "Patch" actions are required to generate new concepts from the one that is aligned to the same span.[12] Ballesteros and Al-Onaizan (2017) uses a DEPENDENT action to generate one tail concept for one hop and cannot deal with the cases which have a chain of more than two concepts aligned to the same span. Our list-based system differs theirs by using a NEW action to deal these cases. Since the new concept is pushed onto the buffer, NEW action can be repeatedly applied and used to generate arbitrary concepts that aligned to the same span. On the development set of LDC2014T12, our oracle achieves 91.7 Smatch F1 score over the JAMR alignment, which outperforms Ballesteros and Al-Onaizan (2017)'s oracle (89.5 in their paper) on the same alignment. This result confirms that our list-based system is more powerful.

[12] e.g., three concepts in the fragment `(person :source (country :name (name :op1 "North" :op2 "Korea")))` are aligned to "*North Koreans*".

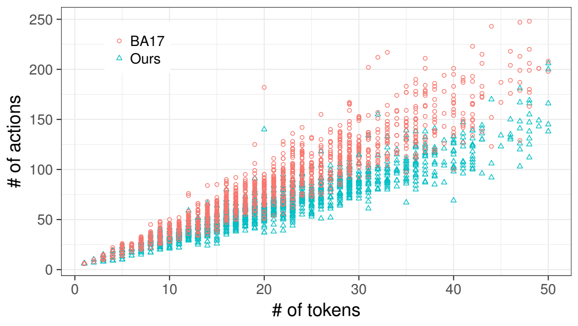

Figure 3: Number of actions required to parse the development set by two systems.

Number of Actions. Our list-based system also differs theirs in the number of oracle actions required to parse the same AMR graphs. We use the oracles from two systems to parse the development set of LDC2014T12 on the same JAMR alignments. Figure 3 shows the comparison in which our system clearly uses fewer actions (the average number of our system is 63.7 and that of Ballesteros and Al-Onaizan (2017) is 86.4). Using fewer actions makes the parser learned from the oracle less prone to error propagation. We attribute the improved performance in Table 6 to this advantage of transition system.

7 Conclusion

In this paper, we propose a new AMR aligner which is tuned by a novel transition-based AMR oracle parser. Our aligner is also enhanced by rich semantic resource and recalls more alignments. Both the intrinsic and extrinsic evaluations show the effectiveness of our aligner by achieving higher alignment F1 score and consistently improving two open-sourced AMR parsers. We also develop transition-based AMR parser based on our aligner and transition system and it achieves a performance of 68.4 Smatch F1 score via ensemble with only words and POS tags as input.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and suggestions. This work was supported by the National Key Basic Research Program of China via grant 2014CB340503 and the National Natural Science Foundation of China (NSFC) via grant 61632011 and 61772153.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proc. of EMNLP*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using Stack-LSTMs. In *Proc. of EMNLP*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proc. of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proc. of ACL*.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proc. of ACL*.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proc. of EACL*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. of ACL*.

Christiane Fellbaum, Anne Osherson, and Peter E. Clark. 2009. Putting semantics into WordNet's "morphosemantic" links. In *Human Language Technology. Challenges of the Information Society*.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss. In *Proc. of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proc. of ACL*.

William Foland and James H. Martin. 2017. Abstract meaning representation parsing using LSTM recurrent neural networks. In *Proc. of ACL*.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing. In *Proc. of ACL*.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and whats next. In *Proc. of ACL*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proc. of ACL*.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proc. of NAACL*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL System Demonstrations*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4).

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for amr parsing. In *Proc. of CoNLL*.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017. Addressing the data sparsity issue in neural amr parsing. In *Proc. of EACL*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP*.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning english strings with abstract meaning representation graphs. In *Proc. of EMNLP*.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing English into abstract meaning representation using syntax-based machine translation. In *Proc. of EMNLP*.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proc. of EMNLP*.

Chuan Wang and Nianwen Xue. 2017. Getting the most out of amr parsing. In *Proc. of EMNLP*.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proc. of ACL*.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for amr parsing. In *Proc. of NAACL*.

Junsheng Zhou, Feiyu Xu, Hans Uszkoreit, Weiguang QU, Ran Li, and Yanhui Gu. 2016. AMR parsing with an incremental joint model. In *Proc. of EMNLP*.