Better Transition-Based AMR Parsing with a Refined Search Space

Zhijiang Guo and **Wei Lu**
Singapore University of Technology and Design
8 Somapah Road, Singapore, 487372
zhijiang_guo@mymail.sutd.edu.sg, luwei@sutd.edu.sg

Abstract

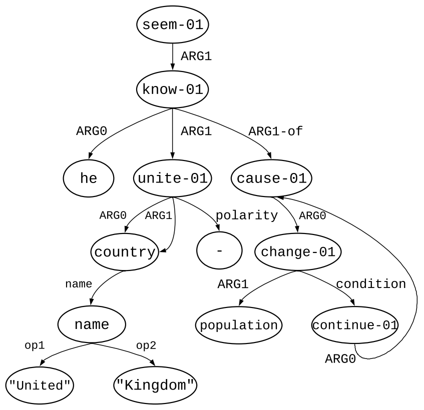
This paper introduces a simple yet effective transition-based system for Abstract Meaning Representation (AMR) parsing. We argue that a well-defined search space for a transition system is crucial for building an effective parser. We propose to conduct the search in a refined search space based on a new compact AMR graph and an improved oracle. Our end-to-end parser achieves the state-of-the-art performance on various datasets with minimal additional information.[1]

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a formalism that captures the semantics of a sentence with a rooted directed graph, in which nodes represent the *concepts* and edges represent the *relations* between concepts. An example AMR graph together with its corresponding sentence are illustrated in Figure 1.

AMR parsing, the task of transforming a sentence into its AMR graph, is a challenging task as it requires the parser to learn to predict not only concepts, which consist of predicates, lemmas, named entities, wiki-links and co-references, but also a large number of relation types based on relatively sparse training data (Peng et al., 2017). Given the challenges, many state-of-the-art AMR parsers employ various external resources and adopt a pipeline approach (Flanigan et al., 2016; Wang and Xue, 2017; Foland and Martin, 2017; van Noord and Bos, 2017). Recently, Damonte et al. (2016); Ballesteros and Al-Onaizan (2017); Peng et al. (2018) have successfully developed AMR parsers in an end-to-end fashion using a transition-based approach.

The transition-based approach (Yamada and Matsumoto, 2003; Nivre, 2003, 2004) has been



It seems that he knows under the population changes (if it continues) that United Kingdom will not unite.

Figure 1: An example AMR graph for a sentence.

popular among many NLP tasks, including syntactic parsing (Zhang and Clark, 2011; Chen and Manning, 2014), named entity recognition (Lample et al., 2016), and semantic parsing (Cheng et al., 2017). Different from the graph-based approach for structured prediction (e.g., conditional random fields (Lafferty et al., 2001)), such an approach is able to maintain a good balance between efficiency and accuracy (Nivre and McDonald, 2008), and has achieved state-of-the-art results on a number of tasks (Swayamdipta et al., 2016; Shi et al., 2017; Cheng et al., 2017).

While the transition-based approach is promising for AMR parsing, existing transition-based AMR parsers still cannot attain the state-of-the-art results on such a task. We observe that the key to the development of an effective transition-based system is a properly defined search space, and argue that the search space used in existing transition systems needs to be refined. Inspired by (Wang et al., 2015), we design a new compact AMR graph representation. Transition actions designed based on such a compact graph enable our parser to generate the target structure with fewer actions and to better capture the correspondence

[1]Dynet (Neubig et al., 2017) is used to implement our parser. We make the supplementary material and code available at http://statnlp.org/research/sp

between concepts and tokens in the sentence.

Oracle, the algorithm used at the training time for specifying the action sequence that can recover the gold AMR graph, is also crucial for the transition-based system. A good oracle will be able to teach the parser how to find a proper path in the search space. The oracle requires alignment information between words and concepts. We identify limitations associated with current practice for finding such alignment information, and propose a new approach that integrates both rules and unsupervised learning.

Experiments show that our system that makes use of POS tags as the only external resource performs competitively on benchmark datasets. To the best of our knowledge, the parser achieves the highest score on the standard LDC2014T12 dataset. Our parser also yields competitive scores on the LDC2015E86 dataset and the more recent LDC2017T10 dataset. On the popular newswire section of LDC2014T12 dataset, our parser outperforms the previous best system by around 3 points in terms of F_1 measure.

2 Related Work

Since the AMR graph encodes rich information, it has been explored for many downstream applications such as language generation (Song et al., 2016), information extraction (Huang et al., 2016) and machine comprehension (Sachan and Xing, 2016). Currently, most parsers can be categorized into 4 classes: 1) *Tree*: such approaches incrementally convert a dependency tree into its corresponding AMR graph (Wang et al., 2015; Goodman et al., 2016; Barzdins and Gosko, 2016); 2) *Graph*: the graph-based models calculate scores of edges and then use a maximum spanning connected subgraph algorithm to select edges that will constitute the graph (Werling et al., 2015; Flanigan et al., 2016); 3) *Seq2seq*: the models adapted from sequence-to-sequence (Sutskever et al., 2014) methods (Peng et al., 2017; Konstas et al., 2017); 4) *Transition*: the transition-based methods, whose input is the plain text sentence and the output is the corresponding graph (Zhou et al., 2016; Damonte et al., 2016; Ballesteros and Al-Onaizan, 2017; Peng et al., 2018).

Apart from these models, Peng et al. (2015) introduce a synchronous hyperedge replacement grammar solution. Pust et al. (2015) regard the task as a machine translation problem, while Artzi et al. (2015) adapt combinatory categorical grammar (Steedman and Baldridge, 2011) for it. Foland and Martin (2017) decompose the parsing task into many subtasks. Then they use multiple bidirectional LSTMs for identifying different types of concepts and relations and iteratively combine these components to form the AMR graph.

Because the mapping between AMR concepts and tokens in the input sentence is latent, external aligners have been developed for training purpose. The most popular aligner is JAMR (Flanigan et al., 2016), which greedily aligns input tokens to graph fragments by using a static template. Another aligner is ISI (Pourdamghani et al., 2014), which is a statistical approach that borrows techniques from statistical machine translation.

3 Approach

We adopt a transition-based approach for AMR parsing. We first propose a new compact representation for AMR graph. Based on our new representation, we further present a novel technique for constructing the action sequence used for training our model. As we will see later, both newly introduced techniques are crucial for building an improved transition-based system within our refined search space.

3.1 Compact AMR Graph

Inspired by (Wang et al., 2015), we design a representation called *compact AMR graph* to simplify concepts and relations of an AMR graph, which makes the learning of our transition system easier. The construction of our compact AMR graph involves removing concepts and relations from an original AMR graph.

Remove Concepts: First we categorize AMR concepts into 2 types:

- **Lexical**: concepts which are converted directly from tokens in the sentence into certain expressions ranging from predicates with sense tags, lemmas to tokens with quotation marks. One example is the concept seem-01 shown in Figure 1, which is converted from the token *seems* with 01 as the sense tag.
- **Non-Lexical**: concepts which are invoked by their child concepts rather than from tokens in the sentence directly. Examples include country and name in Figure 1.[2]

A non-lexical concept is invoked by its child concepts, while a lexical concept corresponds to

[2]A list of non-lexical concepts are provided in the supplementary material.

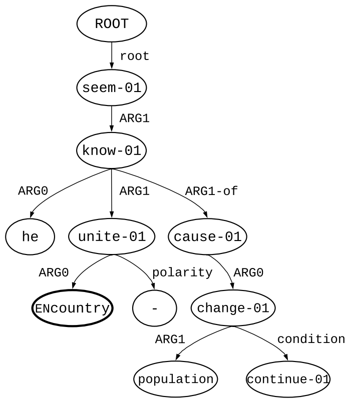


Figure 2: The compact AMR graph for the example in Figure 1.

a certain token in the input sentence. Inspired by (Lu et al., 2008) which learns a semantic parser by aligning words and semantic units, we compress a subgraph rooted by a non-lexical concept into a new concept, which directly corresponds to a contiguous span in the sentence.

For example, a subgraph that consists of non-lexical concepts: country, name and lexical concepts "United", "Kingdom" in Figure 1 can be compressed into one non-lexical concept ENcountry, which can be aligned to the span *"United Kingdom"* in the input sentence as shown in Figure 2.

Removing concepts in the graph enables the parser to build it with fewer actions. Empirically we find that 9% fewer actions are required in LDC2015E86 on average. Also, all concepts can be invoked by a contiguous span in the sentence, which helps the parser to capture the correspondence between concepts and tokens better.

Remove Relations: We also refine the search space by eliminating certain relations in the original AMR graph. The number of relation types is relatively large in the AMR corpus, which leads to a large search space. In order to introduce *reentrancy*[3], attached concepts have to stay in the stack rather than being removed as in most of transition-based AMR parsers.

Take cause-01 shown in the Figure 1 as an example, which is a reentrancy node headed by know-01 and continue-01. After it has been attached to its parent know-01 and child change-01, it still needs to stay in the stack and wait for another relation headed by continue-01. These two concepts are potentially far away from each other in the input sentence, which means cause-01 has to stay in the stack for a long time. In general, the longer a word has to wait to get assigned the more opportunities there are for something to go awry (Manning and Schütze, 1999).

During the testing phase, the parser does not know whether cause-01 is a reentrancy node. Therefore, the parser needs to add all possible relation attachment actions (nearly half of the action space) into the valid action set as long as cause-01 exists in the stack, which makes the parser harder to train.

Thus, we define several properties that the compact graph should respect to further refine the search space[4]:

- **Acyclicity**: cycles are forbidden. The relation ARG0 between cause-01 and continue-01 shown in Figure 1 is removed in the compact AMR graph Figure 2.
- **Simple**: for each parent-child concept pair, only one relation is attached. There exist two relations ARG0 and ARG1 between concepts unite and country shown in Figure 1. One of them is removed in Figure 2.
- **Non-terminal restricted**: only a subset of concepts are allowed to have children. For example, lexical concept – in Figure 1 can only be a terminal node in the compact AMR graph, which means that it can be removed from the stack once it has been attached.
- **Reentrancy restricted**: reentrancy is forbidden for certain concepts, which means these concepts only have one parent concept. An example is that lexical concepts – cannot be reentrancy in the compact AMR graph.

After incorporating these constraints, relation attachment actions are forbidden at many states, which refines the search space. Even though such constraints might prevent us from generating some valid AMR graphs, in practice we can convert the compact AMR graph back to the AMR graph without much loss[5].

3.2 Oracle

Another key to successful search is the oracle, an algorithm that produces a sequence of actions that lead to the gold AMR graph. The action sequence generated by the oracle is significant as it tells

[3]One concept can participate in multiple relations as concept country in Figure 1.

[4]In practice, we impose constraints on the valid action set of the parser for each state to ensure the growing structure always respect these properties. Details are provided in supplementary material.

[5]95% of graphs in the training set of LDC2015E86 dataset satisfy these constraints, which means no relation needs to be removed for most of the graphs.

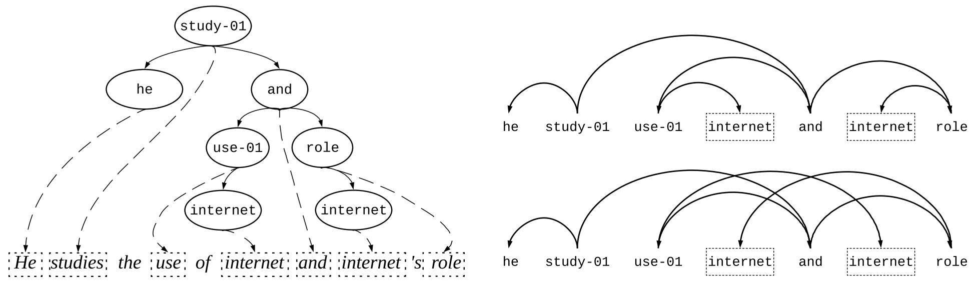

study-01
he
and
use-01
role
internet
internet
He studies the use of internet and internet 's role
he study-01 use-01 internet and internet role
he study-01 use-01 internet and internet role

Figure 3: An alignment example for a sentence that has two identical tokens *internet* and the relation arcs between concepts if we put them on a semi-plane.

the model which actions should be taken during training. The oracle requires, as input, the alignment between tokens in the sentences and concepts in the graph. Almost all prior transition-based parsers have used the JAMR aligner for identifying such alignment information. However, we find that the JAMR aligner suffers from two issues, which may lead to errors that may be propagated to the parser through the oracle.

Non-Projectivity Caused by Alignment Error: Consider a sentence that has two identical tokens–*internet*. Figure 3 illustrates the alignments between concepts and tokens. Since our parser predicts the concepts from left to right, we can draw edges between aligned concepts on a semi-plane to verify whether it has crossing arcs.

If the alignment is correct, the AMR graph is projective as shown at the top right of Figure 3. However, the JAMR is a rule-based aligner, which uses a set of heuristics to do fuzzy matching. Its matching mechanism tends to make mistakes when there exist multiple identical words. Such alignment errors may lead to crossing arcs that our parser may not be able to handle.

Assume an alignment error occurs as illustrated in the bottom right of Figure 3. Such an error will force the parser to attach relations between wrong concepts. Sometimes these concepts are far from each other, which potentially leads to more crossing arcs. For example, ARG1 is attached to the wrong internet, causing the undesired crossing arc.

Empty Alignments:Another issue is that as the corpus gets larger, the JAMR aligner yields more empty alignments as shown in Table 3. Since the set of matching rules is static, unseen patterns in a larger corpus may result in sub-optimal alignment information.

Hybrid Aligner: In order to address these issues, we first try the ISI aligner (Pourdamghani et al., 2014). According to preliminary results (Table 3), we find that the performance of such an unsupervised model is not as good as the JAMR aligner when aligning relation and non-lexical concepts. Therefore, we propose a hybrid aligner, which combines unsupervised learning and rule-based strategy[6].

JAMR does not consider information about the structure whereas the unsupervised models can capture *locality* (Wang and Xue, 2017) – the assumption that words that are adjacent in the source sentence tend to align to words that are closer in the target sentence (Vogel et al., 1996). Structural information can also be incorporated into the model to allow it to capture locality beyond linearity (Wang and Xue, 2017). This property of the unsupervised aligner can alleviate the problem of non-projectivity caused by alignment error. Similar to what is done in the JAMR aligner, we also design rules based on properties of AMR graphs to improve alignments of non-lexical concepts.

In the preprocessing stage of the hybrid aligner, we remove all relations. As non-lexical concepts can be aligned to their child concepts in the compact graph, we then remove all non-lexical concepts. Our unsupervised method is based on IBM word alignment models (Brown et al., 1993).

In the postprocessing stage, we align non-lexical concepts iteratively to the same span that its child concepts are aligned to. For example, non-lexical concepts country and name shown in Figure 1 are removed in preprocessing. During postprocessing, their alignments come from child concepts “United” and “Kingdom”, they are aligned to the 16th token *“United”* and the 17th token *“Kingdom”* respectively. Therefore, non-lexical concepts country and name can be aligned to the span 16-17, which is *“United Kingdom”*.

3.3 Transition System

The transition system consists of a stack S containing words that have been processed, a buffer

[6]Details of the rules and how to apply them in the hybrid aligner are provided in the supplementary material.

Action	$State_t \rightarrow State_{t+1}$
SHIFT	$(S, u\|B) \rightarrow (u\|S, B)$
REDUCE	$(u\|S, B) \rightarrow (S, B)$
RIGHTLABEL(r)	$(u\|S, v\|B) \rightarrow (g_r(u, v, r)\|S, v\|B)$
LEFTLABEL(r)	$(u\|S, v\|B) \rightarrow (u\|S, g_r(v, u, r)\|B)$
SWAP	$(u, v\|S, B) \rightarrow (v, u\|S, B)$
MERGE	$(u\|S, v\|B) \rightarrow (S, g_m(u, v)\|B)$
PRED(n)	$(S, u\|B) \rightarrow (S, n\|B)$
ENTITY(ℓ)	$(S, u\|B) \rightarrow (S, g_\ell(u, \ell)\|B)$
GEN(n)	$(S, u\|B) \rightarrow (S, u, n\|B)$

Table 1: Definition of actions. $(u|S)/(u, v|S)$: item u (or u, v) is at the top of the stack. g_r, g_m and g_ℓ represent composition functions described in Section 4.3.

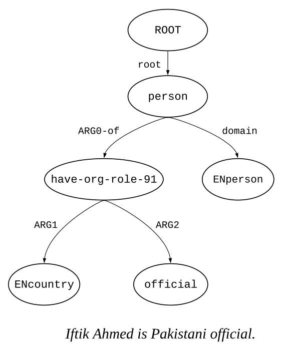


Figure 4: An example sentence with its compact AMR graph.

B containing words to be processed. Initially, S_1 is empty and B_1 contains the whole input sentence and a end-of-sentence symbol at the end. Execution ends on time step t such that B_t is empty and S_t contains a single structure.

Motivated by (Henderson et al., 2013; Ballesteros and Al-Onaizan, 2017), we design 9 types of actions summarized in Table 1. An example for parsing a sentence into its compact AMR graph shown in Figure 4 is provided in Table 2.

- **SHIFT**: removes an item from the front of the buffer and pushes it to the stack.
- **REDUCE**: pops the item on top of the stack.
- **RIGHTLABEL(r)**: creates a relation arc from the item on top of the stack to the item at the front of the buffer. Since these two items are not removed, they can be attached by another relation arc in the future. Therefore, reentrancy is allowed.
- **LEFTLABEL(r)**: creates a relation in the reverse direction as RIGHTLABEL.
- **SWAP**: swaps the top two items on the stack. This action allows non-projective relations (Nivre, 2009) and helps to introduce reentrancy. Repetitive SWAP actions are disallowed to avoid infinite swapping.
- **PRED(n)**: predicts the predicate and lemma concepts corresponding to the item at the front of the buffer. This action requires a look-up table M generated during the training phase. For example, *meet* is mapped to concepts such as meet, meet-01 and meet-02 in M. If the token *meet* is at the front of the buffer, then we add all its corresponding concepts based on M to the valid action sets.
- **MERGE**: removes the item at the front of the buffer and the item on top of the stack, then a composition function is applied to merge them into a new item. The item will be pushed back to the front of the buffer. This action serves to generate non-lexical concepts, whose corresponding span is larger than two. For example, if *"Iftik"* is on top of the stack and *"Ahmed"* is at the front of the buffer, a MERGE action will be applied. This action can be applied recursively if the span is larger than two.
- **ENTITY(ℓ)**: pops the item at the front of the buffer, and labels it with an entity label. This action is designed for named entities in non-lexical concepts. For example, after merging *"Iftik"* and *"Ahmed"*, ENTITY(ENperson) will be used to label the item.
- **GEN(n)**: creates a non-lexical concept invoked by the item at the front of the buffer, which can be any type of concept or composed representation after MERGE transition. Then the non-lexical concept is pushed back to the buffer right after the item. This action can be applied recursively. An example is that concept have-org-role-91 is invoked by *official*. If token *official* is at the front of the buffer, GEN(have-org-role-91) will be applied.

4 Stack LSTMs

A classifier is required to decide which action to take at each time step, given the current state.

4.1 Stack LSTMs for AMR parsing

Stack LSTMs (Dyer et al., 2015) are LSTMs (Hochreiter and Schmidhuber, 1997) that allow stack operations. Using stack LSTMs, each state can be represented using the contents of the stack, buffer and a list with the history of actions.

Let $\mathbf{s}_t$, $\mathbf{b}_t$ and $\mathbf{a}_t$ denote the summaries of stack, buffer and the history of actions at time step t respectively. The parser state $\mathbf{y}_t$ is given by:

$$\mathbf{y}_t = \max\{\mathbf{0}, \mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\} \quad (1)$$

Action	**Stack**	**Buffer**	**Concept/Relation**
		Iftik, Ahmed, is, Pakistani, official, $	-
SHIFT	*Iftik*	*Ahmed, is, Pakistani, official, $*	-
MERGE		*(Iftik, Ahmed), is, Pakistani, official, $*	-
ENTITY (ENperson)		ENperson, *is, Pakistani, official, $*	ENperson
SHIFT	ENperson	*is, Pakistani, official, $*	-
SHIFT	ENperson, *is*	*Pakistani, official, $*	-
ENTITY (ENcountry)	ENperson, *is*	ENcountry, *official, $*	ENcountry
REDUCE	ENperson	ENcountry, *official, $*	-
SHIFT	ENperson, ENcountry	*official, $*	-
GEN (person)	ENperson, ENcountry	*official*, person, *$*	person
GEN (have-org-role-91)	ENperson, ENcountry	*official*, have-org-role-91, person, *$*	have-org-role-91
PRED (official)	ENperson, ENcountry	official, have-org-role-91, person, *$*	official
SHIFT	ENperson, ENcountry, *official*	have-org-role-91, person, *$*	-
LEFTLABEL (ARG2)	ENperson, ENcountry, *official*	have-org-role-91, person, *$*	have-org-role-91$\xrightarrow{\text{ARG2}}$official
REDUCE	ENperson, ENcountry	have-org-role-91, person, *$*	-
LEFTLABEL (ARG1)	ENperson, ENcountry	have-org-role-91, person, *$*	have-org-role-91$\xrightarrow{\text{ARG1}}$ENcountry
REDUCE	ENperson	have-org-role-91, person, *$*	-
SHIFT	ENperson, have-org-role-91	person, *$*	-
LEFTLABEL (ARG0-of)	ENperson, have-org-role-91	person, *$*	person$\xrightarrow{\text{ARG0-of}}$have-org-role-91
SWAP	have-org-role-91, ENperson	person, *$*	-
LEFTLABEL (domain)	have-org-role-91, ENperson	person, *$*	person$\xrightarrow{\text{domain}}$ENperson
REDUCE	have-org-role-91	person, *$*	-
SHIFT	have-org-role-91, person	*$*	-
PRED (ROOT)	have-org-role-91, person	ROOT	ROOT
LEFTLABEL (root)	have-org-role-91, person	ROOT	ROOT$\xrightarrow{\text{root}}$person
REDUCE	have-org-role-91	ROOT	-
REDUCE		ROOT	-
SHIFT	ROOT		-

Table 2: Example action sequence for parsing the sentence: *Iftik Ahmed is Pakistani official.* Here *$* is the end-of-sentence symbol.

where $\mathbf{W}$ is a learned parameter matrix, and $\mathbf{d}$ is a bias term. To learn a better representation, we apply the attention mechanism (Bahdanau et al., 2014) to the words the buffer contains. After getting the representation, we can use it to compute the probability of the next action:

$$p(z|\mathbf{y}_t) = \frac{\exp(g_{z_t}^\top \mathbf{y}_t + q_{z_t})}{\sum_{z' \in \mathcal{A}} \exp(g_{z'}^\top \mathbf{y}_t + q_{z'})} \quad (2)$$

where g_z is a vector representing the embedding of the action z and q_z is a bias term. The set $\mathcal{A}$ represents the valid action set in each time step. This set varies for different parsing states due to the constraints we made for the compact AMR graph.

4.2 Representations and UNK strategy

Following (Dyer et al., 2015), we concatenate three vector representations: pretrained word vector (Ling et al., 2015), learned word vector and learned vector for the POS tag, followed by a linear map to get the representation of each input token. For relation label representations and generated concepts, we simply use the learned embedding of the parser action that was applied to construct the relation.

Dyer et al. (2015) show that these representations can deal flexibly with out-of-vocabulary (OOV) words[7]. We extend this UNK strategy to AMR parsing. Apart from stochastically replacing (with $p = 0.2$) each singleton in the training data, we also replace the original PRED(n) as PRED(*UNK*) if the token should be selected as a concept. At the postprocessing stage, if an AMR concept is generated by PRED(*UNK*) we will use its lemma form for nouns or the most frequent sense for verbs to replace the "UNK" placeholder. This strategy makes the classifier trained not only to predict whether an OOV word should be selected as an AMR concept but also to predict the correct concept for the in-vocabulary word.

4.3 Composition Functions

We use recursive neural networks (Socher et al., 2013) to compute the representations of partially constructed structures.

For relation attachments, a composition function g_r is used to combine representations of AMR parent concept ($\mathbf{h}$), child concept ($\mathbf{d}$) and the corresponding relation label ($\mathbf{r}$) as:

$$g_r(\mathbf{h}, \mathbf{d}, \mathbf{r}) = \tanh(\mathbf{U}_r[\mathbf{h}; \mathbf{d}; \mathbf{r}] + \mathbf{b}_r) \quad (3)$$

where $\mathbf{U}_r$ and $\mathbf{b}_r$ are parameters in the model, as $\mathbf{U}_m$, $\mathbf{b}_m$, $\mathbf{U}_\ell$ and $\mathbf{b}_\ell$ in equation (4) and (5).

For generated non-lexical concepts, if terminal concepts are lexical concepts as shown in Figure 4 we will use a composition function g_m to get their ($\mathbf{c}_1$ and $\mathbf{c}_2$) merged representation:

$$g_m(\mathbf{c}_1, \mathbf{c}_2) = \tanh(\mathbf{U}_m[\mathbf{c}_1; \mathbf{c}_2] + \mathbf{b}_m) \quad (4)$$

[7]Both OOV words in the parsing data and OOV words in the pretraining language model can be represented.

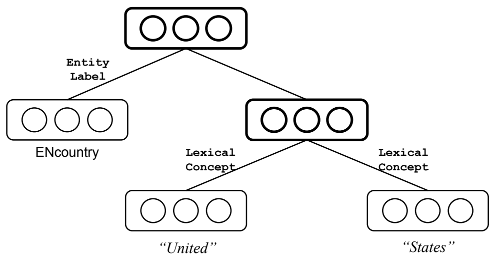


Figure 5: Example of a partial structure rooted by a non-lexical concept. *"United"* and *"States"* are merged by g_m. Then g_ℓ is applied to combine the merged representation and the label representation of ENcompany to arrive at the final representation.

Next, another composition function g_ℓ is applied to get the representation of the labeled generated concept ($\mathbf{e}$ and its entity label $\mathbf{l}$):

$$g_\ell(\mathbf{e}, \mathbf{l}) = \tanh(\mathbf{U}_\ell[\mathbf{e}; \mathbf{l}] + \mathbf{b}_\ell) \qquad (5)$$

5 Experiments and Results

We first evaluate the effectiveness of our hybrid aligner. Then we report the final results by incorporating it into our parser. We conduct experiments on the datasets LDC2014T12, LDC2015E86 and LDC2017T10. The newswire section of LDC2014T12 is popular so we also report the performance on it. All AMR parsing results are evaluated by using Smatch (Cai and Knight, 2013).

5.1 Aligner Evaluation

As the hybrid aligner is designed to tackle non-projectivity caused by alignment error and null alignment, our experiments focus on these aspects.
Non-Projectivity Issue: From Table 3, we find that the percentage of graphs that our hybrid aligner cannot recover is significantly less on these datasets when compared to the use of the JAMR and ISI aligners. We find that the percentage drops less on LDC2017T10. The reason behind this is that many AMR concepts do not have alignments (11.1%) if we use JAMR aligner. The probability of crossing arcs drops owing to the decrease of identified concepts. The ISI aligner shares the same issue. It yields the lowest crossing arcs rate since lots of concepts are not aligned.
Empty Alignment Issue: On the other hand, our hybrid aligner yields less empty alignments. This potentially helps our parser to achieve better performance, since it predicts lexical concepts solely relying on the look-up table generated by the aligner. If we do not have alignment between a certain token and its concept on the training set, the parser is less likely to predict that token as a concept during the testing phase.

Dataset	Aligner	F_1(%)	CR(%)	Null(%)
2014N	JAMR	92.70	14.4	5.1
	Hybrid	**96.12**	**8.8**	**2.0**
2014	JAMR	89.99	16.9	7.0
	Hybrid	**95.27**	**10.5**	**2.6**
2015	JAMR	86.23	16.5	8.9
	Hybrid	**93.05**	11.9	**2.9**
	ISI	71.92	**3.4**	20.1
2017	JAMR	84.41	13.6	11.1
	Hybrid	**92.69**	10.2	**3.4**
	ISI	75.31	**3.2**	18.2

Table 3: Alignment results evaluation. F_1 indicates the final score obtained by the action sequence generated by the aligner during training process. CR(%) indicates the portion that the parser cannot handle because of non-projectivity. Null(%) indicates the percentage of AMR concepts that are not aligned. As the ISI alignments did not include in the LDC2014T12, we can not compare on this dataset.

Improvements on these two aspects allow the aligner to yield better action sequences for the purposes of training our parser. From Table 3, we can see that if we follow the action sequence generated by the hybrid aligner, our parser will achieve a higher Smatch score consistently. Improvement gain gets larger as the corpus gets larger.

5.2 Parser Evaluation

Then we evaluate our parser on the same datasets. As illustrated in Table 4, most models incorporate additional features such as dependency trees, named entities, non-lexical role labels and external corpora. As stack LSTMs use POS tags to get the representation of the input, our parser does not use external resources other than POS tags obtained by using NLTK (Loper and Bird, 2002). We also try to evaluate our parser by removing the POS tags. The performance drops around 1.2 F_1 points on average. POS tags help the parser to select the correct sense when PRED action is applied.
Comparison with other parsers: Currently, all state-of-the-art models either have a relatively high complexity or adopt a pipeline approach. Wang and Xue (2017) incorporate a module called Factor Concept Labels consisting of Bi-LSTMs and CNN based on CAMR (Wang et al., 2016). Another popular parser is JAMR (Flanigan et al., 2016). The relation identification stage of JAMR has the complexity of $O(|V|^2 \log |V|)$, where $|V|$ is the number of concepts. RIGA (Barzdins and Gosko, 2016) is an ensemble system that combines CAMR and seq2seq model. Johnson et al. (2018) view AMR graph as the structure *AM alge-*

Parser	Type	Features					Pipeline	$F_1(\%)$			
		POS	**DEP**	**NER**	**SRL**	**Other**		**2014N**	**2014**	**2015**	**2017**
Flanigan et al. (2014)	Graph	√	√	×	×	No	Yes	59	58	-	-
Flanigan et al. (2016)	Graph	√	√	√	×	No	Yes	-	66	67	-
Werling et al. (2015)	Graph	√	√	√	×	No	Yes	62	-	-	-
Artzi et al. (2015)	Others	√	×	×	×	No	Yes	67	-	-	-
Pust et al. (2015)	Others	×	×	√	×	Word	Yes	-	67.1	-	-
Zhou et al. (2016)	Transition	√	√	√	√	No	No	71	66	-	-
Damonte et al. (2016)	Transition	√	√	√	√	No	No	-	64	64	-
Goodman et al. (2016)	Tree	√	×	√	×	No	Yes	70	-	64	-
Barzdins and Gosko (2016)	Tree	√	√	√	×	No	Yes	-	-	67.2	-
Wang et al. (2015)	Tree	√	√	√	×	No	Yes	70	66.5	-	-
Wang et al. (2016)	Tree	√	√	√	√	No	Yes	-	66.5	67.3	-
Wang and Xue (2017)	Tree	√	√	√	√	No	Yes	-	68.1	68.1	-
Peng et al. (2017)	Seq2seq	×	×	×	×	No	No	-	-	52	-
Konstas et al. (2017)	Seq2seq	×	×	√	×	Giga	No	-	-	62.1	-
Ballesteros and Al-Onaizan (2017)	Transition	√	√	×	×	No	No	69	64	-	-
Foland and Martin (2017)	Others	×	×	√	×	No	Yes	-	-	70.7±0.2	-
Buys and Blunsom (2017)	Seq2seq	√	×	√	×	No	No	-	-	-	61.9
van Noord and Bos (2017)	Seq2seq	√	×	×	×	Silver	No	-	-	68.5	71.0
Peng et al. (2018)	Transition	√	√	×	×	No	No	-	-	64	-
Vilares and Gómez-Rodríguez (2018)	Transition	√	√	√	×	No	No	-	-	64	-
Lyu and Titov (2018)	Others	√	×	√	×	No	Yes	-	-	**73.7±0.2**	**74.4±0.2**
Johnson et al. (2018)	Tree	√	×	√	×	No	No	-	-	70.2±0.3	71.0±0.5
This work (full model)	Transition	√	×	×	×	No	No	**74.0±0.5**	**68.3±0.4**	68.7±0.3	69.8±0.3
no compact AMR graph	Transition	√	×	×	×	No	No	72.1	66.7	67.2	68.9
JAMR aligner	Transition	√	×	×	×	No	No	72.6	65.4	65.8	66.3
no compact AMR graph, *JAMR* aligner	Transition	√	×	×	×	No	No	70.3	63.9	64.6	65.3
no UNK strategy	Transition	√	×	×	×	No	No	72.2	66.4	67.2	68.8
no POS	Transition	×	×	×	×	No	No	72.4	66.9	67.3	68.8

Table 4: Comparison with other parsers on LDC2014T12, LDC2015E86 and LDC2017T10 datasets. 2014N refers to the newswire section of LDC2014T12. We categorize parsers based on their types as discussed in Section 2. We also list down the features used by each system. POS: POS tags; DEP: dependency trees; NER: named entities; SRL: semantic role labels. Other features include: Word (WordNet for concept identification), GIGA (20M unlabeled Gigaword), and Silver (100k additional training pairs created by using CAMR and JAMR). "JAMR aligner" indicates that the parser is trained by action sequence generated by the JAMR aligner. "no UNK strategy" shows results without our UNK strategy. "no compact AMR graph" shows results without constraints on the compact AMR graph. Following several previous work, we also report standard deviation for the full model.

bra defined in (Groschwitz et al., 2017). Since *AM algebra* can be viewed as dependency trees over the sentence, they can train a dependency parser to map the sentence into this structure. Different from the structure used in CAMR (Wang et al., 2015), this structure can be directly transformed to AMR graph by using postprocessing rather than relying on another transition-based system. The complexity of their projective decoder is $O(n^5)$, where n is the length of the sentence. Zhou et al. (2016) is also a transition-based parser, which adopts and improves beam search strategy. In contrast, our system does not use a beam, but we anticipate improved results when a beam is used.

For seq2seq models, they generally require less features and build the AMR graph in an end-to-end way. However, these models usually suffer from data sparsity issue (Peng et al., 2017). In order to address this issue, these models utilize external corpora. Konstas et al. (2017) achieves 62.1 score by using 20M unlabeled Gigaword sentences for paired training. van Noord and Bos (2017) use an additional training corpus of 100k sentences called Silver generated by an ensemble system consisting of JAMR and CAMR parsers. Without training on this additional dataset, the performance of their model is 64.0 on LDC2015E86.

Our end-to-end parser has *linear* time complexity and it exhibits competitive results on the datasets with only POS tags as additional features. Foland and Martin (2017) only use named entities as an external resource and they report the second highest F_1 on LDC2015E86. Their system also adopts a pipeline approach. The concept identification phase requires 5 different LSTMs to discover different kinds of concepts based on carefully designed features. Then they connect these components into a single graph. Unlike previous work, the very recent work by Lyu and Titov (2018) treat the alignments as latent variables in a joint probabilistic model, which improves the parsing performance substantially. They report the highest scores on LDC2015E86 and LDC2017T10 datasets. Their parser requires 5 different BiLSTMs for concept identification, alignment prediction, relation identification and root identification. Though our parser constructs the AMR graph in an end-to-end fashion, it can achieve 68.7 and 69.8 Smatch score respectively on the same test set with a simple architecture.

Metric	LDC2015E86						LDC2017T10			
	W'15	**F'16**	**D'16**	**V'18**	**J'18**	**Ours**	**vN'17**	**L'18**	**J'18**	**Ours**
Smatch	67	67	64	64	**70**	69	71	**74**	71	70
Unlabeled	69	69	69	68	**73**	72	74	**77**	74	73
No WSD	64	68	65	65	**71**	69	72	**76**	72	71
Reentrancies	41	42	41	44	**46**	**46**	**52**	**52**	49	49
Concepts	80	**83**	**83**	**83**	**83**	**83**	82	**86**	**86**	84
Named Ent.	75	79	**83**	**83**	79	80	79	**86**	78	80
Wikification	0	**75**	64	70	71	68	65	**76**	71	70
Negations	18	45	48	47	**52**	49	**62**	58	57	48
SRL	60	60	56	57	**63**	61	66	**70**	64	63

Table 5: Detailed results for the LDC2015E86 and LDC2017T10 test set. W'15 is Wang et al. (2015)'s parser. F'16 is Flanigan et al. (2016)'s parser, D'16 is Damonte et al. (2016)'s parser. V'18 is Vilares and Gómez-Rodríguez (2018)'s parser. J'18 is Johnson et al. (2018)'s parser. vN'17 is van Noord and Bos (2017)'s parser with 100K additional training pairs. L'18 is Lyu and Titov (2018)'s parser.

In order to investigate how our parser performs on predicting *Named Entities*, *Reentrancies*, *Concepts*, *Negations*, etc, we also use the fine-grained evaluation tool (Damonte et al., 2016) and compare to systems which reported these scores. The results are shown in Table 5. We obtain relatively high results for *Concepts*, *Named Entities* and *Wikification*. Also, we achieve good performance on reentrancy identification though we remove many reentrant edges during training. This indicates that the compact AMR graph encodes necessary information. On the other hand, our model does not perform so well on predicating *Negations*. We believe one reason is that our parser is a word-level model. It is hard for it to capture morphological features such as prefixes "un", "in", "il", etc. We anticipate better performance when the character-based representations are used.

Does the compact AMR graph help: In order to better investigate the effect of different modules used in the parser, we also evaluate our parser by removing certain modules. Because our transition system is built based on the compact AMR graph, we could not evaluate the parser by isolating this representation completely. Therefore, we choose to remove some constraints defined on the compact AMR graph to investigate its effect. We can see that our representation improves the performance of our parser on three datasets, which indicates that a refined search space is beneficial to our system. When our parser is trained on the action sequence generated by the JAMR aligner as previous models, it still achieves a competitive score especially on the newswire section.

Impact of the hybrid aligner: Experiments also illustrate that the hybrid aligner consistently helps our parser. When we train our parser by using the action sequence generated by the JAMR aligner, we still achieve competitive results on the newswire section. However, the performance drops as the corpus gets larger. The largest drop occurs when we apply the parser on LDC2017T10, on which correct senses of predicates cannot be selected for many unseen words. We hypothesize that it is because the quality of alignment degrades as the corpus gets larger, which prevents the parser from learning how to find a good path in the search space during training. The hybrid aligner can alleviate this issue by generating a look-up table that has broader coverage. After incorporating the hybrid aligner, our parser achieves the best results of 68.3 on the full test set of LDC2014T12.

6 Conclusion and Future Work

We present a novel transition-based system which refines the search space. Experiments show that our parser is able to achieve state-of-the-art performance with a simple architecture and minimal additional resources. We believe our end-to-end system is helpful in practical settings. In the future, we would also like to investigate if it is possible for alignments to be treated as latent variables, which can be learned in a joint manner within the current framework.

Acknowledgments

We would like to thank the anonymous reviewers for their thoughtful comments. We would also like to thank Zhiyang Teng, Zhanming Jie, Bailin Wang, Aldrian Obaja Muis, Miguel Ballesteros, Swabha Swayamdipta, Bill Yuchen Lin and Yujian Wu for their helpful suggestions. This work is supported by Singapore Ministry of Education Academic Research Fund (AcRF) Tier 2 Project MOE2017-T2-1-156.

References

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

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*.

Guntis Barzdins and Didzis Gosko. 2016. Riga at SemEval-2016 task 8: Impact of Smatch extensions and character-level neural translation on AMR parsing accuracy. In *Proc. of NAACL-HLT*.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proc. of ACL*.

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

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. of EMNLP*.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Proc. of ACL*.

Marco Damonte, Shay B Cohen, and Giorgio Satta. 2016. 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*.

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 SemEval*.

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. UCL+Sheffield at SemEval-2016 task 8: Imitation learning for AMR parsing with an alpha-bound. In *Proc. of SemEval*.

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In *Proc. of IWCS*.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Computational linguistics*, 39(4):949–998.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Lifu Huang, Taylor Cassidy, Xiaocheng Feng, Heng Ji, Clare R Voss, Jiawei Han, and Avirup Sil. 2016. Liberal event extraction and event schema induction. In *Proc. of ACL*.

Mark Johnson, Alexander Koller, Jonas Groschwitz, Meaghan Fowlie, and Matthias Lindemann. 2018. AMR dependency parsing with a typed semantic algebra. 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*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proc. of NAACL-HLT*.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of Word2Vec for syntax problems. In *Proc. of NAACL-HLT*.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028.

Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proc. of EMNLP*.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proc. of ACL*.

Christopher D Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proc. of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proc. of AFNLP*.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. *Proc. of ACL*.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980*.

Xiaochang Peng, Daniel Gildea, and Giorgio Satta. 2018. AMR parsing with cache transition systems. In *Proc. of AAAI*.

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*.

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*.

Mrinmaya Sachan and Eric Xing. 2016. Machine comprehension using rich semantic representations. In *Proc. of ACL*.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proc. of EMNLP*.

Richard Socher, John Bauer, Christopher D Manning, et al. 2013. Parsing with compositional vector grammars. In *Proc. of ACL*.

Linfeng Song, Yue Zhang, Xiaochang Peng, Zhiguo Wang, and Daniel Gildea. 2016. AMR-to-text generation as a traveling salesman problem. In *Proc. of EMNLP*.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and explicit models of grammar*, pages 181–224.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proc. of NIPS*.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2016. Greedy, joint syntactic-semantic parsing with Stack LSTMs. In *Proc. of SemEval*.

David Vilares and Carlos Gómez-Rodríguez. 2018. A transition-based algorithm for unrestricted AMR parsing. In *Proc. of NAACL-HLT*.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proc. of the 16th conference on Computational linguistics-Volume 2*.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. CAMR at SemEval-2016 task 8: An extended transition-based AMR parser. In *Proc. of SemEval*.

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

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

Keenon Werling, Gabor Angeli, and Christopher Manning. 2015. Robust subgraph generation improves abstract meaning representation parsing. In *Proc. of ACL*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational linguistics*, 37(1):105–151.

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*.