AMR Parsing with an Incremental Joint Model

Junsheng Zhou[†], **Feiyu Xu**[‡], **Hans Uszkoreit**[‡], **Weiguang Qu**[†], **Ran Li**[†] and **Yanhui Gu**[†]
† Language Information Processing and Social Computing Lab
School of Computer Science and Technology, Nanjing Normal University, China
{zhoujs, wgqu, gu}@njnu.edu.cn, liran3277@sina.com
‡ Language Technology Lab, DFKI, Germany
{feiyu, uszkoreit}@dfki.de

Abstract

To alleviate the error propagation in the traditional pipelined models for Abstract Meaning Representation (AMR) parsing, we formulate AMR parsing as a joint task that performs the two subtasks: concept identification and relation identification simultaneously. To this end, we first develop a novel component-wise beam search algorithm for relation identification in an incremental fashion, and then incorporate the decoder into a unified framework based on multiple-beam search, which allows for the bi-directional information flow between the two subtasks in a single incremental model. Experiments on the public datasets demonstrate that our joint model significantly outperforms the previous pipelined counterparts, and also achieves better or comparable performance than other approaches to AMR parsing, without utilizing external semantic resources.

1 Introduction

Producing semantic representations of text is motivated not only by theoretical considerations but also by the hypothesis that semantics can be used to improve many natural language tasks such as question answering, textual entailment and machine translation. Banarescu et al. (2013) described a semantics bank of English sentences paired with their logical meanings, written in Abstract Meaning Representation (AMR), which is rapidly emerging as an important practical form of structured sentence semantics. Recently, some literatures reported some promising applications of AMR. Pan et al. (2015) presented an unsupervised entity linking system with AMR, achieving the performance comparable to the supervised state-of-the-art. Liu et al. (2015) demonstrated a novel abstractive summarization framework driven by the AMR graph that shows promising results. Garg et al. (2016) showed that AMR can significantly improve the accuracy of a biomolecular interaction extraction system compared to only using surface- and syntax-based features. Mitra and Baral (2016) presented a question-answering system by exploiting the AMR representation, obtaining good performance.

Automatic AMR parsing is still in a nascent stage. Flanigan et al. (2014) built the first AMR parser, JAMR, based on a pipelined approach, which breaks down the whole task into two separate subtasks: concept identification and relation identification. Considering that node generation is an important limiting factor in AMR parsing, Werling et al. (2015) proposed an improved approach to the concept identification subtask by using a simple classifier over actions which generate these subgraphs. However, the overall architecture is still based on the pipelined model.

As a common drawback of the staged architecture, errors in upstream component are often compounded and propagated to the downstream prediction. The downstream components, however, cannot impact earlier decision. For example, for the verb "affect" in the example shown in Figure 1, there exist two possible concepts: "affect-01" and "affect-02". Comparatively, the first concept has more common use cases than the second one. But, when the verb "affect" is followed by the noun "ac-

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 680–689,
Austin, Texas, November 1-5, 2016.

cent", it should evoke the concept "affect-02". Obviously, the correct concept choice for the verb "affect" should exploit a larger context, and even the whole semantic structure of the sentence, which is more probable to be unfolded at the downstream relation identification stage. This example indicates that it is necessary to allow for the interaction of information between the two stages.

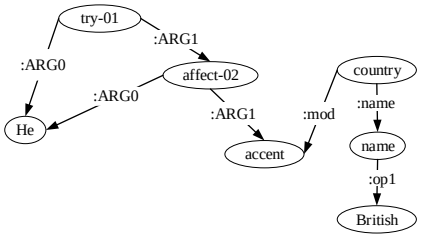

Figure 1: The AMR graph for the sentence "He tries to affect a British accent."

To address this problem, in this paper we reformulate this task as a joint parsing problem by exploiting an incremental parsing model. The underlying learning algorithm has shown the effectiveness on some other Natural Language Processing (NLP) tasks, such as dependency parsing and extraction of entity mentions and relations (Collins and Roark, 2004; Hatori et al., 2012; Li and Ji, 2014). However, compared to these NLP tasks, the AMR parsing is more challenging in that the AMR graph is more complicated. In addition, the nodes in the graph are latent.

One main challenge to search for concept fragments and relations incrementally is how to combine the two subtasks in a unified framework. To this end, we first develop a novel Component-Wise Beam Search (CWBS) algorithm for incremental relation identification to examine the accuracy loss in a fully incremental fashion compared to the global fashion in which a sequence of concept fragments derived from the whole sentence are required as input, as the MSCG algorithm in JAMR. Secondly, we adopt a segment-based decoder similar to the multiple-beam algorithm (Zhang and Clark, 2008b) for concept identification, and then incorporate the CWBS algorithm for relation identification into this framework, combining the two subtasks in a single incremental model. For parameter estimation, "violation-fixing" perceptron is adopted since it is designed specifically for inexact search in structured learning (Huang et al., 2012).

Experimental results show that the proposed joint framework significantly outperforms the pipelined counterparts, and also achieves better or comparable performance than other AMR parsers, even without employing external semantic resources.

2 Background

2.1 AMR Parsing Task

Nodes of an AMR graph are labeled with concepts, and edges are labeled with relations. Concepts can be English words ("He"), PropBank event predicates ("try-01", "affect-02"), or special keywords ("British"). For example, "affect-02" represents a PropBank roleset that corresponds to the first sense of "affect". According to (Banarescu et al., 2013), AMR uses approximately 100 relations. The rolesets and core semantic relations (e.g., ARG0 to ARG5) are adopted from the PropBank annotations in OntoNotes. Other semantic relations include "mode", "name", "time", "topic" and so on. The AMR guidelines provide more detailed descriptions.

2.2 The Pipelined Models for AMR Parsing

The AMR parser JAMR is a two-stage algorithm that first identifies concepts and then identifies the relations that obtain between these.

The concept identification stage maps spans of words in the input sentence to a sequence of concept graph fragments. Note that these graph fragments, in some cases, are subgraphs with multiple nodes and edges, not just one labeled concept node. The relation identification stage adds edges among the concept subgraph fragments identified in the first stage. JAMR requires the output subgraph $G =< V_G, E_G >$ should respect the following constraints:

(1) Simple: For any two vertices u and $v \in V_G$, E_G includes at most one edge between u and v.

(2) Connected: G must be weakly connected (every vertex reachable from every other vertex, ignoring the direction of edges).

(3) Deterministic: For each node $u \in V_G$, and for each label $l \in \{\text{ARG0}, \ldots, \text{ARG5}\}$, there is at

most one outgoing edge in E_G from u with label l.

To find a maximum spanning AMR graph, JAMR proposed a two-step approach[1] . First, a graph that ignores constraint (3) but respects the others was created, by searching for the maximum spanning connected subgraph from an edge-labeled, directed graph representing all possible relations between the identified concepts; Second, a Lagrangian relaxation was adopted to iteratively adjust the edge scores so as to enforce constraint (3).

In order to train the parser, JAMR built an automatic aligner that uses a set of rules to greedily align concepts to spans of words in the training data to generate an alignment table.

3 Algorithms

Based on the hypothesis that concept identification and relation identification are interrelated, we propose to jointly perform the two subtasks in a single model. To this end, we present an incremental model for AMR parsing. Evidence from psycholinguistic research also suggests that human language comprehension is incremental. Comprehenders do not wait until the end of the sentence before they build a syntactic or semantic representation for the sentence.

However, the challenges of successfully applying the incremental joint model to this problem formulation are: 1) how can we design an effective decoding algorithm for identifying the relations between the nodes in an incremental fashion, given a partial sequence of spans, i.e., a partial sequence of gold-standard concept fragments; 2) further, if given a sentence, how can we design an incremental framework to perform concept identification and relation identification simultaneously. In the following subsections we introduce our solutions to these challenges in detail.

3.1 An Incremental Decoding Algorithm for Relation Identification

We define the relation identification problem as finding the highest scoring graph y from all possible outputs given a sequence of concept fragments c:

$$F(c) = \arg\max_{Gen(c)} Score(y) \quad (1)$$

where $Gen(c)$ denotes the set of possible AMR graph for the input c. The score of an output parse y is defined to be decomposed by edges, and with a linear model:

$$Score(y) = \sum_{e \in E_y} w^T \cdot \varphi(e) \quad (2)$$

where $\varphi(e)$ is the feature vector over the edge e, and w is weight vector of the model.

The AMR graph is a directed graph that respects three constraints (see section 2.2) and has a node marked as the focus node. Obviously, finding such a maximum spanning graph in AMR parsing in fact carries more complexities than that of maximum spanning tree (MST) decoding for syntactic parsing. Especially, performing the task incrementally is substantially harder than doing it non-incrementally. In both cases, parsing is in general intractable and we provide an approximate inference algorithm to make these cases tractable.

Inspired by the graph-based dependency parser under the framework of beam-search, which yields a competitive performance compared to the exact-search-based counterpart (Zhang and Clark, 2008a), we develop a CWBS algorithm for the relation identification task.

Basically, the decoder works incrementally, building a state item (i.e. a partial AMR graph) fragment by fragment. When each concept fragment is processed, edges are added between the current concept fragment and its predecessors. However, how to treat its predecessors is a difficult problem. In our experiments, we found that if we consider every preceding concept fragment to the left of the current fragment in a right-to-left order in the search process, the decoder suffers from low efficiency and poor performance. Unlike the beam-search for dependency parsing, which can greatly reduce the search space by exploiting the projectivity property of the dependency tree (Covington, 2001; Zhang and Clark, 2008a), this naive search process in this context inevitably leads to huge search space, and furthermore is difficult to guarantee the connectivity of

[1] In this paper, we refer to this two-step approach for relation identification as MSCG algorithm.

output graph. Instead, we propose a component-wise beam search scheme, which can not only alleviate much noisy partial candidate, but also ensure that the final output graph is *connected*.

Algorithm 1 shows the pseudocode for the complete procedure of the decoder. In a nutshell, the algorithm builds the AMR graph in one left-to-right pass over the sequence of concept fragments. Beam search is applied by keeping the B-best[2] items in the agenda at each processing stage, according to the scores of partial graph up to the current concept fragment. Lets take an illustrative diagram to demonstrate the procedure (see Figure 2). When appending the current concept fragment to the left partial graph to extend it, we just need to consider the relations between current concept and each preceding connected component. However, even at this single step, picking B-best extended partial graphs is still a difficult task due to the large combination space. Here, we adopt an effective *nested beam search strategy* at this step. In other words, edges are added between the current concept fragment and its preceding connected components by iterating through these components in a right-to-left order[3] using an *inner beam-search*. When examining the edges between the current concept fragment and some preceding component, four elementary actions are used:

(1) **SHIFT** (lines 12-14): Add only current concept to the partial graph.

(2) **LEFT-ARC** (lines 16-19): Add current concept and a highest-scoring edge from a node in the current concept to a node in some preceding connected component to the partial graph.

(3) **RIGHT-ARC** (lines 21-24): Add current concept and a highest-scoring edge from a node in some preceding connected component to a node in current concept to the partial graph.

(4) **LEFT & RIGHT-ARCS** (lines 26-27): Add current concept and highest-scoring left arc and right arc to the partial graph.

The first three actions are similar in form to those in the Arc-Standard algorithm for transition-based dependency parsing (Nivre, 2008; Zhang and Clark, 2008a). The last one is defined to cope with the cases where there may be multiple parents for some nodes in an AMR graph. Note that the "SHIFT" action does not add any edges. This operation is particularly necessary because the partial graphs are not always connected during the search process. In our experiments, we also found that the number of connected components during search process is relatively small, which is generally less than 6. It is important to note that, in order to guarantee the output graph *connected*, when the last concept fragment is encountered, the "SHIFT" action is skipped (see line 10 in Algorithm 1), and the other three 'arc' actions will add edges to connect the last concept fragment with all preceding connected components to yield a connected graph.

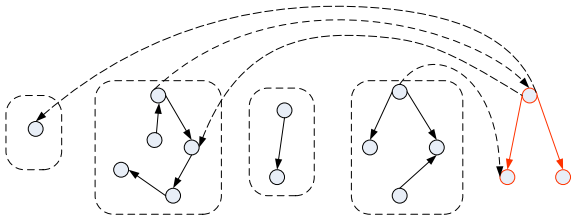

Figure 2: An illustrative diagram for CWBS algorithm. Each dotted box corresponds to a connected component in the partial graph, each of which consists one or multiple concept fragments. The rightmost subgraph corresponds to the current concept fragment.

For purpose of brevity, we introduce some functional symbols in Algorithm 1. Function CalEdgeScores($state, c_i$) calculates the scores of all candidate edges between the nodes in current concept fragment c_i and the nodes in the partial graph in state covering $(c_1, c_2, \ldots, c_{i-1})$. For computing the scores of edges, we use the same features as JAMR (refer to Flanigan et al. (2014) for more details). Function FindComponents($state$) returns all connected components $(p_1, p_2, \ldots, p_m)$ in the partial graph in $state$, sorted by the maximum end position of spans including in every component. The AddItem function adds the current concept fragment and left/right arc to the partial graph. Function AppendItem($buf, item$) inserts the partial graph $item$ into buf by its score. Functions GetMaxLeftEdge(c_i, p_j) and

[2]The constant B denotes the beam size.

[3]The right-to-left order reflects the principle of local priority.

Algorithm 1 The incremental decoding algorithm for relation identification.

Input: A sequence of concept fragments $(c_1, c_2, \ldots, c_n)$
Output: Best AMR graph including $(c_1, c_2, \ldots, c_n)$

$agenda \leftarrow$ {Empty-graph}
for $i \leftarrow 1 \ldots n$ **do**
  for $state$ in $agenda$ **do**
    CalEdgeScores($state$, c_i)
    $(p_1, p_2, \ldots, p_m) \leftarrow$ FindComponents($state$)
    $innerAgenda \leftarrow state$
    for $j \leftarrow m \ldots 1$ **do**
      $buf \leftarrow NULL$
      for $item$ in $innerAgenda$ **do**
        if $i < n$ **then**
          //Add only c_i to the item
          $newitem \leftarrow item$
          AddItem($newitem, c_i$)
          AppendAgenda($buf, newitem, i, n$)
        // Add a left arc from c_i to p_j to the $item$
        $newitem \leftarrow item$
        $le \leftarrow$ GetMaxLeftEdge(c_i, p_j)
        AddItem($newitem, c_i, le$)
        AppendAgenda($buf, newitem, i, n$)
        //Add a right arc from p_j to c_i the $item$
        $newitem \leftarrow item$
        $re \leftarrow$ GetMaxRightEdge(p_j, c_i)
        AddItem($newitem, c_i, le$)
        AppendAgenda($buf, newitem, i, n$)
        //Add both left and right arc to the $item$
        AddItem($item, c_i, le, re$)
        AppendAgenda($buf, item, i, n$)
      $innerAgenda \leftarrow B$-best(buf)
  $agenda \leftarrow innerAgenda$
return $agenda$[0]
function AppendAgenda($buf, item, i, n$)
  //parameter n represents the terminal position
  if $i = n$ **then**
    CalRootFeatures($item$)
  AppendItem($buf, item$)

GetMaxRightEdge(p_j, c_i) pick the highest-scoring left-arc and right-arc linking current fragment c_i and the connected component p_j by the scores returned from the CalEdgeScores function, respectively.

Finally, the function CalRootFeatures(g) first computes the scores for all nodes in the output graph g by treating them as the candidate root respectively, and then pick the node with the highest score as the focus node of the graph. When computing the score for each candidate node, similar to JAMR, two types of features were used: the concept of the node, and the shortest dependency path from a word in the span to the root of the dependency tree.

The time complexity of the above algorithm is $O(MB^2n)$, where M is the maximum number of connected components during search, B is beam size and n is the number of concept fragments. It is linear in the length of sequence of concept fragments. However, the constant in the O is relatively large. In practice, the search space contains a large number of invalid partial candidates. Therefore, we introduce three partial output pruning schemes which are helpful in reducing search space as well as making the input for parameter update less noisy.

Firstly, we limit the number of children and parents of every node. By observing the training data, we set the maximum numbers of children and parents of every node as 7 and 4, respectively. Secondly, due to the fact that all frame arguments ARG0-ARG5 are derived from the verb framesets, the edges with label $l \in \{ARG0, \ldots, ARG5\}$ that do not outgo from a verb node will be skipped.

Finally, consider the determinism constraint (as illustrated in section 2.2) that should be satisfied by an AMR representation. When one edge has the same label $l \in \{ARG0, \ldots, ARG5\}$ as one of edges outgoing from the same parent node, this edge will also be skipped. Obviously, this type of pruning can enforce the *determinism* constraint for every decoding output.

3.2 Joint Decoding for Concept Identifica-tion and Relation Identification

In this section, we further consider the joint decoding problem for a given sentence x, which maps the sentence x to an output AMR graph y. The objective function for the joint decoding is as follows:

$$\hat{y} = \arg\max_{y' \in Gen(x)} (w^T \cdot \phi(x, y') + w^T \cdot \mathbf{f}(y')) \quad (3)$$

where the first term is to calculate the score over all concept fragments derived from the words in the sentence x, and the second one is to calculate the score over all edges linking the concept fragments. Maximizing Equation (3) amounts to concurrently maximizing the score over the concept fragments and the score over the edges. Admittedly, the joint decoding problem is more intricate and in general

intractable. Therefore, we use a beam-search-based incremental decoder for approximate joint inference during training and testing.

In order to combine the two subtasks in a unified framework, we first relax the exact-search for concept identification in JAMR by beam search, resulting in a segment-based decoder similar to the multiple-beam algorithm in (Zhang and Clark, 2008b; Li and Ji, 2014), and then incorporate the CWBS algorithm for relation identification (as depicted in section 3.1) into this framework, which provides a natural formulation for combining the two subtasks in a single incremental model.

Algorithm 2 shows the joint decoding algorithm. In short, during performing joint decoding incrementally for the input sentence, for each word index i in the input sentence, it maintains a beam for the partial graphs whose last segments end at the i-th word, which is denoted as $agendas[i]$ in the algorithm. When the i-th word is processed, it either triggers concepts starting from this word by looking up the alignment table generated from the training data, or evokes no concept (we refer to this type of words as *function words*). If the current word triggers multiple concepts, we first append each candidate concept to the partial graphs in the beam $agendas[i-1]$, by using a component-wise beam search way (see section 3.1), and then pick B-best extended partial graphs by exploiting the features from *both the concept level* and *relation level* to compute the overall scores.

In particular, judging whether a word is a *function word* is an important and difficult task. For example, the word "*make*" corresponds to multiple candidate concepts in the alignment table, such as "*make-01*" and "*make-02*". However, it can also act as a functional word in some cases. To resolve the judgement problem, we view each word as a *function word* and a *non-function word* at the same time to allow them to compete against each other by their scores. For instance, for the i-th word, this is done by combining all partial graphs in the beam $agendas[i-1]$ with those in the beam $agendas[i]$ to select B-best items and then record them in $agendas[i]$, which is represented as the Union function in Algorithm 2.

After all words are processed, the highest-scoring graph in the beam corresponding to the terminal position of the sentence is selected as the output.

Algorithm 2 The joint decoding algorithm.

Input: Input sentence $x = (w_1, w_2, \ldots, w_n)$
Output: Best AMR graph derived from x

```
agendas[0] ← ∅
last ← Scan(x)
for i ← 1 ... n do
    list ← Lookup(x, i)
    if list.size > 0 then
        preAgenda ← agendas[i − 1]
        for cf ∈ list do
            end ← i + cf.size − 1
            if preAgenda.size = 0 then
                g ← Graph.empty
                CalConceptFeatures(g, cf)
                AppConcept(agendas, end, g, cf, last)
            else
                for item ∈ preAgenda do
                    g ← item
                    CalConceptFeatures(g, cf)
                    AppConcept(agendas, end, g, cf, last)
        Union(agendas, i, i − 1)
    else
        agendas[i] ← agendas[i − 1]
bestGraph ← agendas[last][0]
return bestGraph
```

In algorithm 2, function Scan(x) is used to search the terminal position corresponding to the last concept fragment in the sentence x, which will be passed as a parameter to the function AppConcept. The Scan function can be efficiently implemented by calling the function Lookup in a right-to-left order. Function Lookup(x, i) maps a sequence of words starting from the index i in sentence x to a set of candidate concept fragments, by looking up the alignment table that was generated from the training data. The alignments are accomplished using an aligner from JAMR. Motivated by Werling et al. (2015), we also adopt two additional actions to generate the candidate concept fragments: LEMMA and VERB. The action LEMMA is executed by using the lemma of the source token as the generated node title, and the action VERB is to find the most similar verb in PropBank based on Jaro-Winkler distance, and adopt its most frequent sense.

Function CalConceptFeatures(g, cf) calculates the feature vector for the candidate concept fragment cf and the partial graph g, using the features

defined in Table 1. Among them, features 1-4 are from JAMR. Additional features 5-16 aim to capture the association between the current concept and the context in which it appears. Function AppConcept($agendas, end, g, cf, last$) appends the current concept cf to the partial graph g, and then inserts the extended partial graph into $agendas[end]$. Note that when the parameter end equals to the parameter $last$, this function will call the function CalRootFeatures to select the focus node, as illustrated in Algorithm 1.

	Name	Description
1	Fragment given words	Relative frequency estimates of the probability of a concept fragment given the span of words.
2	Span length	The length of the span.
3	NER	1 if the span corresponds to a named entity, 0 otherwise.
4	Bias	1 for any concept fragment from the alignment table, 0 otherwise.
5	c	c represents the current concept label, w represents the current words, lem represents the current lemmas, pos represents the current POS tags. w_{-1} denotes the first word to the left of current word, w_{+1} denotes the first word to the right of current word, and so on.
6	$c + w$	
7	$c + lem$	
8	$c + pos$	
9	$c + w_{-1}$	
10	$c + w_{+1}$	
11	$c + pos_{-1}$	
12	$c + pos_{+1}$	
13	$c + w_{-2}$	
14	$c + w_{+2}$	
15	$c + pos_{-2}$	
16	$c + pos_{+2}$	

Table 1: Features associated with the concept fragments.

3.3 Violation-Fixing Perceptron for Training

Online learning is an attractive method for the structured learning since it quickly converges within a few iterations (Collins, 2002). Particularly, Huang et al. (2012) establish a theoretical framework called "violation-fixing perceptron" which is tailored for structured learning with inexact search and has provable convergence properties. Since our incremental decoding for AMR parsing is an approximate inference, it is very natural to employ violation-fixing perceptron here for AMR parsing training.

Specifically, we use an improved update method "max-violation" which updates at the worst mistake, and converges much faster than early update with similar or better accuracy. We adopt this idea here as follows: decode the whole sentence, and find the word index i^* where the difference between the candidate partial graph and gold-standard one is the biggest. Only part of the graph ending at the word index i^* is used to calculate the weight update, in order to account for search errors.

To reduce overfitting, we used averaged parameters after training to decode test instances in our experiments. The resulting model is called averaged perceptron (Collins, 2002).

Additionally, in our training algorithms, the implementation of the oracle function is rela-tively straightforward. Specifically, when the i-th span is processed in the incremental parsing process, the partial gold-standard AMR graph up to the i-th span consists of the edges and nodes that appear before the end position of the i-th span, over which the gold-standard feature vectors are calculated.

4 Experiments

4.1 Dataset and Evaluation Metric

Following previous studies on AMR parsing, our experiments were performed on the newswire sections of LDC2013E117 and LDC2014T12, and we also follow the official split for training, development and evaluation. Finally, we also show our parsers performance on the full LDC2014T12 dataset. We evaluate the performance of our parser using Smatch v2.0 (Cai and Knight, 2013), which counts the precision, recall and F1 of the concepts and relations together.

4.2 Development Results

Generally, larger beam size will increase the computational cost while smaller beam size may reduce the performance. As a tradeoff, we set the beam size as 4 throughout our experiments. Figure 3 shows the training curves of the averaged violation-fixing perceptron with respect to the performance on the both development sets. As we can see the curves converge very quickly, at around iteration 3.

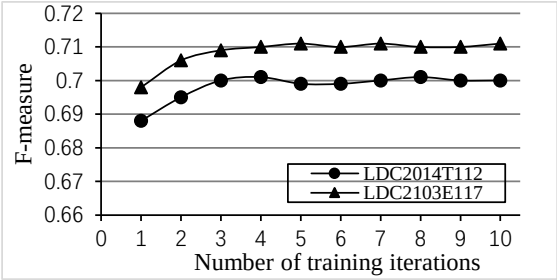

Figure 3: Learning curves on development sets.

Dataset	System	P	R	F1
LDC2013E117	MSCG	.85	.77	.81
	CWBS	.85	.78	.81
LDC2014T12	MSCG	.84	.77	.80
	CWBS	.84	.77	.80

Table 2: Results of two different relation identification algorithms.

4.3 Incremental Relation Identification Performance

Before performing joint decoding, we should first verify the effectiveness of our incremental algorithm CWBS. The first question about CWBS is whether the component-wise search is a valid scheme for deriving the gold-standard AMR graph given the sequence of gold-standard concepts. Therefore, we first implement an oracle function by performing the incremental component-wise search for each fragment sequence c to get a "pseudo-gold" graph $G_c^{'}$; Then we compare with gold-standard AMR graph G_c . On the training data of LDC2013E117 and LDC2014T12, we respectively got an overall 99.6% and 99.7% F-scores for all $< G_c^{'}, G_c >$ pairs, which indicates that our component-wise search is an effective incremental search scheme.

Further, we train a perceptron model using the max-violation update to approximate the oracle search procedure. As shown in Table 2, our incremental algorithm CWBS achieves almost the same performance as the non-incremental algorithm MSCG in JAMR, using the same features as MSCG. The results indicate that CWBS is a competitive alternative to MSCG.

4.4 Joint Model vs. Pipelined Model

In this section, we compare the overall performance of our joint model to the pipelined model, JAMR[4]. To give a fair comparison, we first implemented ***system 1*** only using the same features (i.e., features 1-4 in Table 1) as JAMR for concept fragments. Table 3 gives the results on the two datasets. In terms of F-measure, we gain a 6% absolute improvement, and a 5% absolute improvement over the results of JAMR on the two different experimental setups respectively.

Next, we implemented ***system 2*** by using more lexical features to capture the association between concept and the context (i.e., features 5-16 in Table 1). Intuitively, these lexical contextual features should be helpful in identifying concepts in parsing process. As expected, the results in Table 3 show that we gain 3% improvement over the two different datasets respectively, by adding only some additional lexical features.

Dataset	System	P	R	F1
LDC2013E117	JAMR(fixed)	.67	.58	.62
	System 1	.72	.65	.68
	System 2	.73	.69	.71
LDC2014T12	JAMR(fixed)	.68	.59	.63
	System 1	.74	.63	.68
	System 2	.73	.68	.71

Table 3: Comparison between our joint approaches and the pipelined counterparts.

Dataset	System	P	R	F1
LDC2013E117	CAMR*	.69	.67	.68
	CAMR	.71	.69	.70
	Our approach	**.73**	**.69**	**.71**
LDC2014T12	CAMR*	.70	.66	.68
	CAMR	.72	.67	.70
	CCG-based	.67	.66	.66
	Our approach	**.73**	**.68**	**.71**

Table 4: Final results of various methods.

4.5 Comparison with State-of-the-art

We give a comparison between our approach and other state-of-the-art AMR parsers, including CCG-based parser (Artzi et al., 2015) and dependency-based parser (Wang et al., 2015b). For comparison

[4]We use the latest, fixed version of JAMR, available at `https://tiny.cc/jamr`.

purposes, we give two results from two different versions of dependency-based AMR parser[5]: CAMR* and CAMR. Compared to the latter, the former denotes the system that does not use the extended features generated from the semantic role labeling system, word sense disambiguation system and so on, which is directly comparable to our system.

From Table 4 we can see that our parser achieves better performance than other approaches, even without utilizing any external semantic resources.

We also evaluate our parser on the full LDC2014T12 dataset. We use the training/development/test split recommended in the release: 10,312 sentences for training, 1,368 sentences for development and 1,371 sentences for testing. For comparison, we include the results of JAMR, CAMR*, CAMR and SMBT-based parser (Pust et al., 2015), which are also trained on the same dataset. The results in Table 5 show that our approach outperforms CAMR*, and obtains comparable performance with CAMR. However, our approach achieves slightly lower performance, compared to the SMBT-based parser, which adds data and features drawn from various external semantic resources.

Dataset	System	P	R	F1
LDC2014T12	JAMR(fixed)	.64	.53	.58
	CAMR*	.68	.60	.64
	CAMR	.70	.62	.66
	SMBT-based	-	-	**.67**
	Our approach	.70	.62	.66

Table 5: Final results on the full LDC2014T12 dataset.

5 Related Work

Our work is motivated by JAMR (Flanigan et al., 2014), which is based on a pipelined model, resulting in a large drop in overall performance when moving from gold concepts to system concepts.

Wang et al. (2015a) uses a two-stage approach; dependency parses are modified by executing a sequence of actions to resolve dis-crepancies between dependency tree and AMR structure. Goodman et al. (2016) improves the transition-based parser with the imitation learning algorithms, achieving almost the same performance as that of Wang et al. (2015b), which exploits the extended features from additional trained analysers, including co-reference and semantic role labelers. Artzi et al. (2015) introduces a new CCG grammar induction algorithm for AMR parsing, combined with a factor graph to model non-compositional phenomena. Pust et al. (2015) adapts the SBMT parsing framework to AMR parsing by designing an AMR transformation, and adding external semantic resources. More recently, Damonte et al. (2016) also presents an incremental AMR parser based on a simple transition system for dependency parsing. However, compared to our parser, their parser cannot parse non-projective graphs, resulting in a limited coverage.

Our work is also inspired by a new computational task of incremental semantic role labeling, in which semantic roles are assigned to incomplete input (Konstas et al., 2014).

6 Conclusions and Future Work

In this paper, we present a new approach to AMR parsing by using an incremental model for performing the concept identification and relation identification jointly, which alleviates the error propagation in the pipelined model.

In future work, we plan to improve the parsing performance by exploring more features from the coreference resolution, word sense disambiguation system and other external semantic resources. In addition, we are interested in further incorporating the incremental semantic role labeling into our incremental framework to allow bi-directional information flow between the two closely related tasks.

Acknowledgments

This research is supported by projects 61472191, 61272221 under the National Natural Science Foundation of China, projects 14KJB520022, 15KJA420001 under the Natural Science Research of Jiangsu Higher Education Institutions of China, and partially supported by the German Federal Ministry of Education and Research (BMBF) through the project ALL SIDES (01IW14002) and BBDC (contract 01IS14013E). We would also like to thank the insightful comments from the three anonymous reviewers.

[5]The code is available at `https://github.com/Juicechuan/AMRParsing`

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG Semantic Parsing with AMR. In *Proc. of EMNLP*, pages 1699–1710.

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 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*, pages 748–752.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proc. of ACL*, pages 111–118.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron algorithms. In *Proc. of EMNLP*, pages 1–8.

Michael A. Covington. 2001. A Fundamental Algorithm for Dependency Parsing. In *Proc. of ACM Southeast Conference*.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2016. An Incremental Parser for Abstract Meaning Representation. arXiv preprint at arXiv:1608.06111.

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*, pages 1426–1436.

Sahil Garg, Aram Galstyan, Ulf Hermjakob, and Daniel Marcu. 2016. Extracting Biomolecular Interactions Using Semantic Parsing of Biomedical Text. In *Proc. of AAAI*.

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*, pages 1–11.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Junichi Tsujii. 2012. Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *Proc. of ACL*, pages 1045–1053.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured Perceptron with Inexact Search. In *Proc. of HLT-NAACL*, pages 142–151.

Ioannis Konstas, Frank Keller, Vera Demberg, and Mirella Lapata. 2014. Incremental Semantic Role Labeling with Tree Adjoining Grammar. In *Proc. of EMNLP*, pages 301–312.

Qi Li and Heng Ji. 2014. Incremental Joint Extraction of Entity Mentions and Relations. In *Proc. of ACL*, pages 402–412.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward Abstractive Summarization Using Semantic Representations. In *Proc. of NAACL*, pages 1086–1077.

Arindam Mitra and Chitta Baral. 2016. Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning. In *Proc. of AAAI*.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Xiaoman Pan, Taylor Cassidy, Ulf Hermjakob, Heng Ji, and Kevin Knight. 2015. Unsupervised Entity Linking with Abstract Meaning Representation. In *Proc. of NAACL*, pages 1130–1139.

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*, pages 1143–1154.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. A Transition-based Algorithm for AMR Parsing. In *Proc. of NAACL*, pages 366–375.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. Boosting Transition-based AMR Parsing with Re-fined Actions and Auxiliary Analyzers. In *Proc. of ACL*, pages 857–862.

Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust Subgraph Generation Improves Abstract Meaning Representation Parsing. In *Proc. of ACL*, pages 982–991.

Yue Zhang and Stephen Clark. 2008a. A Tale of Two Parsers: Investigating and Combining Graph-Based And transition-Based Dependency Parsing Using Beam-search. In *Proc. of EMNLP*, pages 562–571.

Yue Zhang and Stephen Clark. 2008b. Joint Word Segmentation and POS Tagging Using a Single Perceptron. In *Proc. of ACL*, pages 888–896.