Learning to Map Dependency Parses to Abstract Meaning Representations

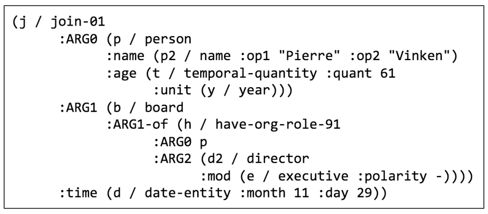
Wei-Te Chen
Department of Computer Science
University of Colorado at Boulder
Weite.Chen@colorado.edu

Abstract

Abstract Meaning Representation (AMR) is a semantic representation language used to capture the meaning of English sentences. In this work, we propose an AMR parser based on dependency parse rewrite rules. This approach transfers dependency parses into AMRs by integrating the syntactic dependencies, semantic arguments, named entity and co-reference information. A dependency parse to AMR graph aligner is also introduced as a preliminary step for designing the parser.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic formalism that expresses the logical meanings of English sentences in the form of a directed, acyclic graph. AMR focuses on the semantic concepts (nodes on the graph), and relations (labeled edges on the graph) between those concepts. AMR relies heavily on predicate-argument structures defined in the PropBank (PB) (Palmer et al., 2005). The representation encodes rich information, including semantic roles, named entities, and co-reference information. Fig. 1 shows an example AMR.



```
(j / join-01
      :ARG0 (p / person
            :name (p2 / name :op1 "Pierre" :op2 "Vinken")
            :age (t / temporal-quantity :quant 61
                  :unit (y / year)))
      :ARG1 (b / board
            :ARG1-of (h / have-org-role-91
                  :ARG0 p
                  :ARG2 (d2 / director
                        :mod (e / executive :polarity -))))
      :time (d / date-entity :month 11 :day 29))
```

Figure 1: The AMR annotation of sentence "Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29."

In this proposal, we focus on the design of an automatic AMR parser in a supervised fashion from dependency parses. In contrast with recent semantic parsing algorithms, we start the parsing process from the dependency parses rather than the sentences. A dependency parse provides both the semantic dependency information for the sentence, and the structure of the relations between the head word and their dependencies. These can provide strong features for semantic parsing. By using a binary-branching bottom-up shift-reduced algorithm, the statistical model for the rewrite rules can be learned discriminatively. Although the AMR parser is my thesis topic, in this proposal we will pay more attention to preliminary work - the AMR -Dependency Parse aligner.

To extract the rewrite rules and the statistical model, we need the links between AMR concepts and the word nodes within the dependency parse. An example alignment is shown in Fig. 2. Alignment between an AMR concept and dependency node is needed because 1) it represents the meaning of the sub-graph of the concept and its child concepts corresponding to the phrase of the head word node, and 2) the dependency node contains sufficient information for the extraction of rewrite rules. For example, the word node "Vinken" on the dependency parse side in Fig. 2 links to the lexical concept "Vinken" and, furthermore, links to the "*p2/name*" and the "*p/person*" concepts since "Vinken" is the head of the named entity (NE) "Pierre Vinken" and the head of the noun phrase "Pierre Vinken, 61 years old." The secondary aim of this proposal is to design an alignment model between AMR concepts and dependency parses. We use EM to search the hidden derivations by combining the features of lexical form, relation label, NE, semantic role, etc. After EM processing, both the alignments and all the feature probabilities can be estimated.

The design of a rewrite-based AMR parser is described in Sec. 2, and the aligner is in Sec. 3. Our preliminary experiments and results are pre-

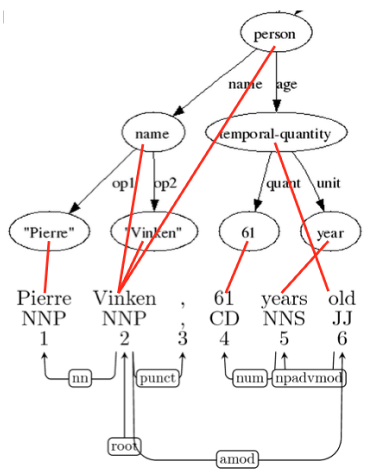


Figure 2: The alignment between an AMR subgraph and a dependency parse. A red line links the corresponding concept and dependency node.

sented in Sec. 4, followed by future work.

2 Rewrite Based AMR Parser

AMR is a rooted, directed, acyclic graph. For example, the concept **join-01** in Fig. 1 is the root meaning of the sentence, which links to the child concepts **Arg0**, **Arg1**, and **time**. AMR adheres to the following principles (Banarescu et al., 2013):

- AMRs are rooted acyclic graphs with labels (relations) on edges. These labels indicate the directed relation between two concepts.
- AMRs abstract away from syntactic idiosyncracies of a language, and instead attempt to capture only the core meaning of a sentence.
- AMRs use the PB framesets as relation labels (Palmer et al., 2005). For example, the relation labels (i.e., ARG0, ARG1) of "join-01" concept in Fig. 1 correspond to the roles of the PB frame "join-v."
- AMRs combine multiple layers of linguistic annotation, like coreference, NE, semantic role, etc., in a single structure.

The above basic characteristics make the parsing of AMRs a difficult task. First, because AMR abstracts away from syntactic idiosyncrasies, we need a model to link the AMR concepts to words in the original sentence, in order to obtain external lexical, syntactic and semantic features. Secondly, the parser should learn the different feature transformation probabilities jointly since AMRs combine several linguistic annotations. Moreover,

(x)/NP $\longrightarrow$:op(x)	-$r1$
(x) nn (y) $\longrightarrow$:name (name(x, y))	-$r2$
(x)/CD $\longrightarrow$:quant(x)	-$r3$
(x)/NNS $\longrightarrow$:unit(x)	-$r4$
npadvmod $(x)(y)$ $\longrightarrow$ temporal-quanity(x, y)	-$r5$
old/JJ (x) $\longrightarrow$:age(x)	-$r6$
NE_PERSON$(x)(y)$ $\longrightarrow$ person(x, y)	-$r7$

Table 1: Sample Rewrite Rules

AMR uses graph variables and reentrancy to express coreference (e.g., "*p*" variable in Fig 1 appears twice – in *:ARG0* of *join-01* and *:ARG0* of *have-org-role-91*). The reentrancy prevents the AMR graph from begin a tree structure. During parsing decoding, a polynomial time algorithm should be replaced by alternative algorithms, like beam search, to avoid an exponential running time.

JAMR (Flanigan et al., 2014) is the first system for AMR parsing, which identifies the concepts, and then searches for a maximum spanning connected subgraph (MSCG) on a fully connected graph to identify the relations between concepts. The search algorithm is similar to the maximum spanning tree algorithms. To assure the final connected graph conforms to linguistic constraints, JAMR uses Lagrangian relaxation (Geoffrion, 2010) to supplement the MSCG algorithm. JAMR reaches a 58% Smatch score (Cai and Knight, 2013) on automatic concept and relation identification data, and 80% on gold concept and automatic relation identification data.

2.1 Our Shift-Reduce Rewrite Rule Parser

Rewrite rule based parser is a bottom-up converter from dependency parses to AMRs. The process starts from the leaf word node on the dependency parse. By applying rewrite-rules to each word node, we obtain and assemble the sub-graphs of our target AMR. Sample rewrite rules are listed in Table 1. In these rules, the left hand side contains the dependency information (e.g. word lemma, POS, relation label, NE tag, etc). The right hand side is the AMR concept and its template for filling variables from previous parsing steps. The sample derivation steps are listed in Table 2. For every step, it shows the derivation rule applied (in Table 1), and the concept name, $c1$-$c8$.

This approach to parsing could be implemented with a shift-reduce algorithm (Wang et al., 2015). We define a stack and a list of tokens, which stores the dependency words in the order of tree traversal. Several actions are defined to operate on the list(L) and the stack(S):

Derivation	**Apply Rule**	**Concept Name**
Pierre/NNP $\longrightarrow$:op Pierre	$r1$	$c1$
Vinken/NNP $\longrightarrow$:op Vinken	$r1$	$c2$
$(c1)$ nn $(c2)$ $\longrightarrow$:name (name :op1 Pierre :op2 Vinekn)	$r2$	$c3$
61/CD $\longrightarrow$:quant 61	$r3$	$c4$
years/NNS $\longrightarrow$:unit year	$r4$	$c5$
npadvmod $(c4)(c5)$ $\longrightarrow$ temporal-quanity :quant 61 :unit year	$r5$	$c6$
old/JJ $(c6)$ $\longrightarrow$:age (temporal-quanity :quant 61 :unit year)	$r6$	$c7$
NE_PERSON $(c3)(c7)$ $\longrightarrow$ person :name (name :op1 Pierre :op2 Vinekn) :age (temporal-quanity :quant 61 :unit year)	$r7$	$c8$

Table 2: The derivation for parsing "Pierre Vinken, 61 years old" from dep. parse to AMR

- **Shift** Remove the dependency word from L, apply the rules, and push the new concept to S.
- **Reduce** Move the two top sub-concepts from S, apply the rules, and push it back to S.
- **Unary** Move the top sub-concept from S, apply the rules, and push it back to S.
- **Finish** If no more dependency words are in the list, and one concept is in S, then return.

The final AMR concept would be stored at the top of the stack. It is guaranteed that all the AMR expressions can be derived from the dependency parses by using the shift-reduce algorithm.

3 Dependency Parses to AMR Aligner

A preliminary step for our rewrite-based parser is the alignment between the AMR and the dependency parse. JAMR (Flanigan et al., 2014) provides a heuristic aligner between an AMR concept and the word or phrase of a sentence. They use a set of aligner rules, like NE, fuzzy NE, data entity, etc., with a greedy strategy to match the alignments. This aligner achieves a 90% F_1 score on hand aligned AMR-sentence pairs. On the other hand, Pourdamghani et al. (2014) present a generative model to align from AMR graphs to sentence strings. They raises concerns about the lack of sufficient data for learning derivation rules. Instead, they propose a string-to-string alignment model, which transfers the AMR expression to a linearized string representation. Then they use several IBM word alignment models (Brown et al., 1993) on this task. IBM Model-4 with a symmetric method reaches the highest F_1 score of 83.1%. Separately analyzing the alignments of roles and non-roles (lexical leaf on AMR), the F_1 scores are 49.3% and 89.8%, respectively.

In comparison to previous work, our aligner estimates the alignments by learning the transformation probability of lexical form, relations, named entities and semantic roles features jointly. Both the alignment and transformation probabilities are initialized for the training of parser.

3.1 Our Aligner Model with EM Algorithm

Our approach, based on the existing IBM Model (Brown et al., 1993), is an AMR-to-Dependency parse aligner, which represents one AMR as a list of Concepts $C = \langle c_1, c_2, \ldots, c_{|C|} \rangle$, and the corresponding dependency parse as a list of dependency word nodes $D = \langle d_1, d_2, \ldots, d_{|D|} \rangle$. The alignment A is a set of mapping functions a, which link Concept c_j to dependency word node d_i, $a : c_j \rightarrow d_i$. Our model adopts an asymmetric EM approach, instead of the standard symmetric one. We can always find the dependency label path between any pair of dependency word nodes. However, the number of concept relation label paths is not deterministic. Thus, we select the alignment direction of AMR to dependency parse only, and one-to-one mapping, in our model.

The objective function is to learn the parameter θ in the AMR-to-Dependency Parse of EM:

$$\theta = \operatorname{argmax} L_\theta(AMR|DEP)$$

$$L_\theta(AMR|DEP) = \sum_{k=1}^{|S|} \sum_{A} P(C^{(k)}, A|D^{(k)}; t, q)$$

where L_θ is the likelihood that we would like to maximize, S is the training data set. We will explain the transformation probability t and the alignment probability q below.

Expectation-Step

The E-Step estimates the likelihood of the input AMR and dependency parse by giving the transformation probability t and alignment probability q. The likelihood can be calculated using:

$$P(A|C, D) = \prod_{j=1}^{|C|} P(c_j|a(c_j))$$

$$P(c_j|d_i, |C|, |D|) = t(c_j|d_i) * q(d_i|c_j, |C|, |D|)$$

We would like to calculate all the probabilities of possible alignments A between c_j and d_i. The transformation probability t is a combination (multiple) probability of several different features:

- $P_{lemma}(c_j|d_i)$: the lemma probability is the probability of the concept name of c_j, conditioned on the dependency word of d_i.

- $P_{rel}(Label(c_j, c_j^p)|RelPath_{dep}(a(c_j), a(c_j^p)))$: the relation probability is the probability of the relation label between c_i and its parent concept c_i^p, given the relation path between the dependency word nodes $a(c_i)$ and $a(c_i^p)$. e.g., the relation probability of $c_j = \mathit{61}$ and $a(c_j) = \underline{61}$ in Fig. 2 is $P(quant|\underline{npadvmod \downarrow num \downarrow})$.
- $P_{NE}(Name(c_j)|Type_{NE}(a(c_j)))$: the NE probability is the probability of the name of c_j, given the NE type (e.g., PERSON, DATE, ORG, etc.) contained by $a(c_j))$.
- $P_{SR}(Label(c_j, c_j^p)|Pred(a(c_j^p)), Arg(a(c_j)))$: the semantic role probability is the probability of relation label between c_j and its parent c_j^p, conditioned on the predicate word of $a(c_j^p)$ and argument type of $a(c_j)$ if $a(c_j)$ is semantic argument of predicate $a(c_j^p)$.

On the other hand, the alignment probability $q(Dist(a(c_j), a(c_j^p))|c_j, |C|, |D|)$ can be interpreted as the probability of the distance between $a(c_j)$ and $a(c_j^p)$ on dependency parse D, conditioned on c_j, the lengths of D and C.

Maximization-Step

In the M-Step, the parameter θ^r is updated from the previous round of θ^{r-1}, in order to maximize the likelihood $L_\theta(AMR|DEP)$:

$$t(C|D; AMR, DEP) = \frac{\sum_{(AMR,DEP)} cnt(C|D; AMR, DEP)}{\sum_C \sum_{(AMR,DEP)} cnt(C|D; AMR, DEP)}$$

$$q(D|C; AMR, DEP) = \frac{\sum_{(AMR,DEP)} cnt(D|C; AMR, DEP)}{\sum_D \sum_{(AMR,DEP)} cnt(D|C; AMR, DEP)}$$

where cnt is the normalized count that is collected from the accumulating probability of all possible alignment from the E-step. EM iterates the E-step and M-step until convergence.

Initialization

Before iterating, the transformation probability t and alignment probability q must be initialized. We use these steps to initialize the parameters:

1. Assign a fixed value, say 0.9, to $P_{lemma}(c_j|d_i)$ if the concept name of c_j is identical or a partial match to the dependency word node d_i. Otherwise, initialize it uniformly;
2. Run the EM algorithm with the initialized P_{lemma} only (Similar to IBM Model 1, which is only concerned with translation probability);
3. Initialize all the other parameters, i.e., P_{rel}, P_{NE}, P_{SR}, and q with the development data;
4. Run the completed EM algorithm with the P_{lemma} we obtained from Step 2 and other probabilities from Step 3.

The extra EM for the initialization of P_{lemma} is to estimate a more reasonable P_{lemma}, and to speed up the convergence of the second round of EM.

Decoding

To find the alignment of $\langle C, D\rangle$, we define the search for alignments as follows:

$$\begin{aligned} &\operatorname*{argmax}_A P(A|C, D) \\ &= \operatorname*{argmax}_A \prod_{j=1}^{|C|} t(c_j|a(c_j)) * q(a(c_j)|c_j, |C|, |D|) \end{aligned}$$

This decoding problem finds the alignment A with the maximum likelihood. A dynamic programming (DP) algorithm is designed to extract the target alignment without exhaustively searching all candidate alignments, which will take $O(|D|^{|C|})$.

This DP algorithm starts from the leaf concepts and then walks through parent concepts. In c_j, we need to produce the following likelihoods:

1. Accumulated likelihood for aligning to any d_i from all the child concepts of c_j
2. Likelihood of P_{lemma} and P_{NE}
3. Likelihood of P_{rel} and P_{SR} for parent concept c_j^p aligned to any dependency word node d_l.

In step (3), we need to find the d_l, aligned by c_j^p, that maximizes the likelihood. The accumulated likelihood is then stored in a list with size=$|D|$. We can trace back and find the most likely alignments in the end. The running time of this algorithm is $O(|C||D|^2)$. This algorithm does not include reentrancy cases. One solution to be explored in future work is to use a beam-search algorithm instead.

4 Preliminary Experiments and Results

Here, we describe a preliminary experiment for the AMR-Dependency Parse aligner, including the data description, experimental setup, and results.

4.1 Data

The LDC AMR release 1.0 consists of 13,051 AMR-English sentence pairs[1]. To match an AMR

[1]LDC AMR release 1.0, Release date: June 16, 2014 https://catalog.ldc.upenn.edu/LDC2014T12

Split	Sent.	Tokens	# of NE	# of Pred.	# of Args
Train	1,000	19,923	1,510	4,231	7,739
Dev.	100	2,328	239	235	526
Test	100	1,672	80	199	445

Table 3: The data split of train/dev./test set. "# of NE", "# of Pred." and "# of Args" stand for the number of named entities, predicate and argument annotations in the data set, respectively.

	P	R	F_1
P_{lemma}	56.7	50.5	53.4
Combination	61.1	53.4	57.0

Table 4: Experiment Results

with its corresponding dependency parse, we select the sentences which appear in the OntoNotes 5.0 release[2] as well, then randomly select 1,000 of them as our training set. The OntoNotes data contains TreeBank, PB, and NE annotations. Statistics about the AMR and OntoNotes corpus and the train/dev./test splits are given in Table 3. We manually align the AMR concepts and dependency word nodes in the dev. and test sets. We initialize P_{rel}, P_{NE}, and P_{SR} with the dev. set.

4.2 Results

We run our first round of EM (Step 2 in Initialization of Sec. 3.1) for 100 iterations, then use the second round (Step 4 in Initialization of Sec. 3.1) for another 100 iterations. We run our decoding algorithm and evaluation on the test set after the first and second round of EM. Due to time constraints, we did not train the q here.

The experimental results are listed in Table 4. We evaluate the performance on the precision, recall, and F_1 score. Using just the P_{lemma} (a similar approach to (Pourdamghani et al., 2014)), we achieve 53.4% F_1 score on the test set. On the other hand, our aligner reaches 57.0% F_1 score with the full aligner.

5 Conclusion and Future Work

In this research, we briefly introduce AMR. We describe the design principles and characteristics of AMR, and show how the AMR parser task is important, yet difficult. We present the basic idea for a proposed AMR parser, based on the shift-reduce algorithm. We also present an AMR-Dependency Parse aligner, because such an aligner will be a necessary first step before parsing. The alignment and the estimated feature probabilities are obtained by running the EM algorithm, which could be use directly for the AMR parser.

In the future, we will be following these steps to develop the proposed rewrite-based parser:

Implemention of our rewrite-based AMR parser: We would like to implement the proposed rewrite-based AMR parser. In comparison to the parser of Flanigan (2014) , we believe our parser could perform better on the runtime. We also plan to experiment with the data generated by an automatic dependency parser.

Expand the experimental data of aligner: One problem discovered in our preliminary experiments was that of data sparsity, especially for P_{lemma}. The LDC AMR Release contains 18,779 AMR/English sentences, and 8,996 of them are contained in the OntoNotes release as well. Therefore, increasing the training data size from the release is one solution to improve the performance of our aligner from the unsatisfactory results. Using external lexical resources, like WordNet, is another promising solution to extend to snyonyms.

Evaluation of the aligner with existing parser: Since our aligner provides the alignment between the dependency word node and both the AMR leaf concept and role concept, we assume that our aligner could improve not only our rewrite-based parser but other parsers as well. To verify this, we hope to submit our improved alignment results to a state-of-the-art AMR parser, and evaluate the parsing results.

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation Grants IIS-1116782, A Bayesian Approach to Dynamic Lexical Resources for Flexible Language Processing, 0910992 IIS:RI: Richer Representations for Machine Translation, and NSF IIA-0530118 PIRE (a subcontract from Johns Hopkins) for the 2014 Frederick Jelinek Memorial Workshop for Meaning Representations in Language and Speech Processing, and funding under the BOLT and Machine Reading programs, HR0011-11-C-0145 (BOLT) FA8750-09-C-0179 (M.R.). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[2]LDC OntoNotes Release 5.0, Release date: October 16, 2013 https://catalog.ldc.upenn.edu/LDC2013T19

References

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.

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

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752. Association for Computational Linguistics.

J. Flanigan, S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proc. of ACL*, Baltimore, Maryland, June. Association for Computational Linguistics.

ArthurM. Geoffrion. 2010. Lagrangian relaxation for integer programming. In Michael Jnger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 243–281. Springer Berlin Heidelberg.

Martha Palmer, Dan Guildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–105, March.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning english strings with abstract meaning representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429. Association for Computational Linguistics.

Chuan Wang, Xue Nianwen, and Pradhan Sameer. 2015. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.