Robust Sub-Graph Generation for Abstract Meaning Representation Parsing

Keenon Werling
Stanford University
keenon@stanford.edu

Gabor Angeli
Stanford University
gabor@stanford.edu

Chris Manning
Stanford University
manning@stanford.edu

Abstract

The Abstract Meaning Representation (AMR) is a representation for open-domain rich semantics Generating semantic sub-graphs from contiguous tokens is a crucial part of AMR parsing. We propose a small set of actions to *construct* a sub-graph at test time from a span of tokens, which allow us to greatly expand our generalization from training data. We show that our set of construction actions is a good approximation which we can learn with a simple classifier. This reduces the need for sparse dictionary lookups, which improves generalization on unknown words and allows us to exploit statistical efficiency on a small training set. We demonstrate that our approach improves on published state-of-the-art AMR parsing, from *0.58* smatch to *0.64* smatch on the LDC2013E117 dataset.

1 Introduction

The Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a rich language for expressing semantic understanding on both a broad domain and at a relatively deep level. AMR captures many useful pieces of semantic information in a single joint representation. These include (but are not limited to) named entity recognition, verb argument labeling, word sense disambiguation, time expression parsing, and coreference. There is an ongoing AMR data labeling effort that promises to produce a breakthrough resource in broad domain semantic parsing, for both its size and the AMR formalism's expressive richness.

As of this writing there is one published parser that targets the AMR formalism, JAMR (Flanigan et al., 2014), which reports very promising results. After experimentation with several different structured prediction algorithms, we find that JAMR's architecture is a very strong framework for further parser development. JAMR follows a two-stage approach to parse AMR, first generating a set of AMR sub-graphs from spans of the sentence, and then stitching those sub-graphs together with a constrained MST (using dual decomposition).

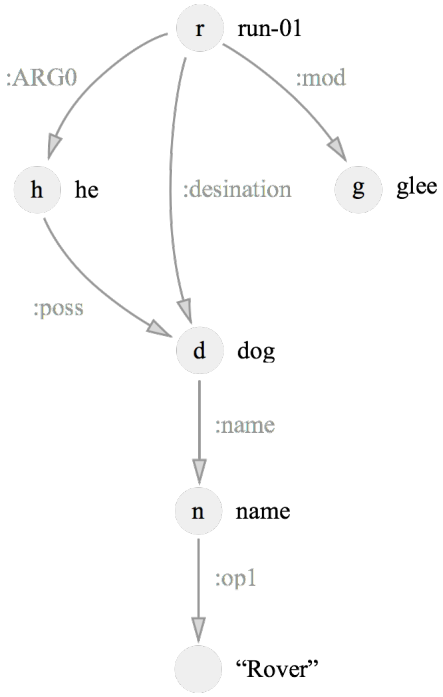

Figure 1: AMR graph for "He gleefully ran to his dog Rover". Nodes represent concepts, and arcs are relationships between concepts.

However, JAMR's architecture, while it has special case machinery for creating sensible chunks for named entities and time expressions, cannot generate AMR sub-graphs from unseen words in the general case.

For instance, the adjective *exhaustive* in "the *exhaustive* search" should be parsed as an AMR verb, *exhaust-01*, but JAMR is unable to recognize this because it has no instance of the adjective *exhaustive* used in the training data. Our system can

generalize from training data to learn that *exhaustive* is to be treated as a verb, and match against the OntoNotes [citation needed] database of verb sense frames at test time to produce *exhaust-01*.

Instead of memorizing a bank of AMR sub-graphs to generate from spans of tokens, we propose a small set of 'generative actions' that our system can take to derive an AMR sub-graph from a span of tokens (see Figure 4). For example, we have an action *IDENT* that will generate a node with the same title as the token in the source sentence that receives the label, which allows us to capture unseen nouns like 'mother'. We show that having a set of actions as an intermediate to generating AMR sub-graphs greatly improves statistical efficiency when training a sub-graph generation system on the relatively small amount of available training data, and that end to end performance in-domain is improved from **0.58** smatch to **0.64** smatch when using JAMR's constrained MST to stitch together the resulting sub-graphs. This approach also enables us to use dense features to improve brittle out-of-domain performance significantly.

2 A Crash-Course in AMR

AMR is a language for expressing semantic understanding that represents meaning as a directed graph, where nodes represent concepts and arcs are relationships between concepts. AMR makes no effort to have a one-to-one correspondence between nodes in a graph and tokens in the sentence whose semantics is being represented. Thus AMR is not a "semantic dependency" representation. AMR represents the relationships between objects referred to by the surface text, not merely the relationships between the words themselves. In fact, AMR will often expand single tokens into large sub-graph elements, or ignore tokens completely.

To introduce AMR and its notation, we'll unpack the translation of the sentence "he gleefully ran to his dog Rover". We show in Figure 1 the interpretation of this sentence as an AMR graph.

Note that the root node of the graph is labeled "run-01". This is the name of a verb sense definition drawn from PropBank [citation needed] for the sense of the verb "ran" in this sentence. This distinguishes this use of the verb "run" from senses like those expressed in "he *ran* the business" or "he gave him a *run* for his money".

Figure 2: AMR representation of the word "sailor", which is notable for breaking the word up into a self-contained multi-node unit

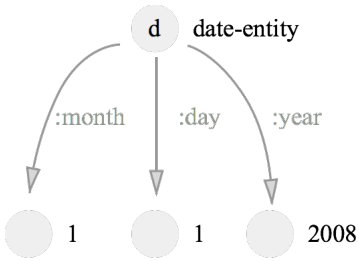

Figure 3: AMR representation of the span "January 1, 2008", an example of how AMR can represent structured data by hallucinating additional nodes like "date-entity" to signify the presence of special structure

"run-01" has an outgoing "ARG0" arc to a node "he", with semantics (drawn from the PropBank frame) that roughly correspond to "he" being the doer of the "run-01" action. The "run-01" has an outgoing "mod" to "glee," which has the catch-all semantics that "run-01" is somehow modified by the concept "glee." "run-01" also has a "destination" arc to "dog," which draws its semantics from Vivek Srikumar's thesis chapter on preposition sense tagging [citation needed], and means that the destination of the "run-01" action is "dog". Then we have a section of the graph that is best interpreted as a unit, where all of the children of "dog" effectively mean that "dog" has the name "Rover."

AMR makes an attempt to capture some semantic meanings in words that are difficult to capture in a way that is not domain specific. For example, the token "sailor" in a sentence will evoke the concept graph representing a person who performs the action "sail-01", see Figure 2. This is difficult to model without resorting to memorization, because the etymological clues are so sparse. We note this as an area for further exploration.

AMR can also capture structured data, like time expressions, see Figure 3. In dates, a "date-entity" node is hallucinated to signify that this cluster of nodes is part of a structured sub-component of an AMR graph, with specific semantics. Dates are a good example of a recurring pattern in AMR, which is to have an "artificial node" signify that all its immediate children are part of a structured piece of data, with some special interpretation. The most common example of this pattern is the "name" node, which signifies that its immediate children comprise the tokens of a name object.

3 Previous Work

At the time of this writing, the JAMR parser (Flanigan et al., 2014) is the only published AMR parser. The crucial insight in JAMR is that AMR parsing can be broken into two relatively distinct tasks: interpreting what entities are being referred to in the text (generating the AMR nodes), and then discovering what relationships those entities have between one another other (linking AMR nodes).

It uses a two-stage approach to parsing AMR. In the first stage, a sequence model is used to generate small AMR sub-chunks. Then in the second stage these chunks are stitched together by a variation of a maximum spanning tree algorithm with dual decomposition to impose linguistically motivated constraints.

To more carefully define what is meant by AMR sub-chunks, **TODO:** write this

4 Methods

Our focus for this paper is on the initial stage of AMR parsing, the generation of AMR sub-graphs from surface text. Our approach is very simple: if you discretize the AMR sub-graph space in terms of the actions you need to take to "derive" a node from its source text, you greatly improve statistical efficiency and domain transfer ability for your parser.

4.1 Derivation Tags

Our method relies on a very simple insight about AMR: a vast majority of the AMR nodes can be explained by a very small set of derivation actions from their source span of text. We do a sequence labeling of each token in the sentence with one of the following action types:

- **VERB**: Look for the most similar PropBank frame, make that the title of the corresponding node.
- **IDENTITY**: Take the lowercased version of the token to be the title of the corresponding node.
- **VALUE**: Parse the token to an integer value, and use that as the node.
- **LEMMA**: Take the lemma of the token to be the title of the corresponding node.
- **NONE**: Ignore this token in the final output.
- **DICT**: Look up the most probable chunk associate with this lexical span. This functions as a back off if no other actions are appropriate.

This has two primary benefits: We're able to more effectively generalize from training data to unseen examples, and we're able to exploit much greater statistical efficiency over our training data.

The only class that doesn't give us an immediate win if we correctly classify it is **DICT**, for which we still have to resort to the JAMR expedient of looking up a span in a dictionary. This class functions as a catch-all. To see how rarely we need to resort to **DICT**, we turn to the distribution of different tag types on the training data, by token.

TAG	# Tokens	% Total
NONE	28405	0.405
DICT	15861	0.226
VERB	11017	0.157
IDENTITY	10890	0.155
LEMMA	2581	0.036
VALUE	710	0.01
COREF	552	0.0078

Table 1: Distribution of tag types in the training data for LDC2013E117 dataset, generated from automatically aligned data.

Note that **DICT** counts for less than 25% of the training data, meaning that more than 75% of tokens can be generated correctly by our tag type classifier even if we've never seen them before, allowing our broad domain parsing to improve dramatically.

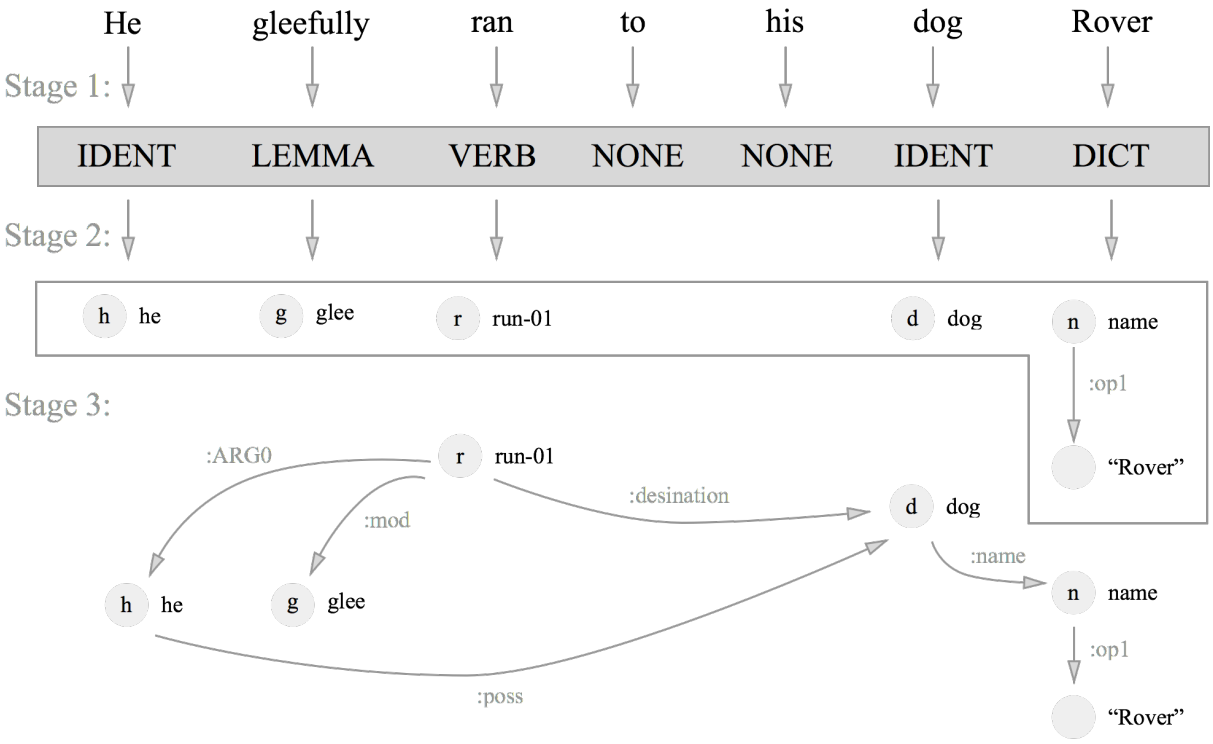

Figure 4: Derivation process for "He gleefully ran to his dog Rover". First the tokens in the sentence are labeled with derivation actions, then those actions are used to generate AMR sub-graphs, and then those sub-graphs are stitched together to form a coherent whole.

The **DICT** class was intended to be used for things that a system cannot know without memorization, like a mapping between "the Lusitania" and a graph describing a ship named Lusitania. One of the reasons that the **DICT** class is so disappointingly large is that it's stealing from **LEMMA**, because AMR will aggressively normalize words and change their part of speech to a semantic equivalent. For example, 'gleefully' gets mapped to 'glee' and not 'gleeful', which is hard to do automatically with stemming rules in the general case.

4.2 Tag Classifier

We use an extremely simple max-ent classifier to make tag decisions. The classifier takes as input a pair $< i, S >$, where i is the index of the token in the input sentence, and S is a sequence tokens representing the source sentence. The output of the classifier is a tag T such that the likelihood with respect to the data of token i in sentence S generating a node according to the action specified by T is maximized. See Appendix A for a list of classifier features.

Our classifier can be analyzed in isolation from the complete pipeline system, which yields useful insights about fundamental difficulties in parsing AMR, and some future directions.

TODO: error analysis

4.3 Chunk Generator

Our algorithm for generating chunks is simple and deterministic. We run our tag classifier for each token the sentence, generating a sequence labeling for the sentence. We then group all spans of adjacent **DICT** tags, and find the maximum likelihood set of generated nodes by doing a Viterbi decoding over all sub-span splits of the adjacent tokens. All other tokens are

TODO: Write out algorithm

4.4 Training Classifier

Our training data comes in the form of a list of (sentence, graph) pairs, so we first generate an 'alignment' between our graphs and the source sentences, in the form of a vector A (see **TODO:** ref sec Automatic Alignments), where $A_i = j$ means that node i is created by some operation on token j. We cluster adjacent nodes with the same source token, and if one token is responsible for multiple disjoint clusters (as sometimes happens in the case of multiple refer-

ences that are not explicitly coreferent, e.g. "Iran and China's *nuclear program*), we take the largest cluster for that token. This yields a set of

5 Automatic Alignment of Training Data

AMR training data is in the form of bi-text, where we are given a set of (sentence,graph) pairs, with no explicit alignments between them. For example, imagine we are given the graph for "He gleefully ran to his dog Rover", as shown in Figure 1. Although it's obvious to a human, the computer has no idea that the node "run-01" came from the token "ran". There is therefore a crucial task of generating these alignments prior to running training algorithms.

To define exactly what is meant by an 'alignment', let there be a pair $G =< N, A >$ where N is a set of nodes and A is an $|N|$ by $|N|$ matrix of binary variables, representing the presence or absence of directed arcs between nodes. For example, $A_{i,j} = T$ means that an arc exists between N_i and N_j, and $A_{i,j} = F$ means that no arc exists between N_i and N_j. Let there be a set of tokens S, such that S_i is the ith token in the source sentence. We would like an array A, where $|A| = |N|$, and for all i, A_i is in the range $(1, |S|)$. For $A_i = n$, it means that token n generated N_i.

In plain english, we want a projective mapping from nodes to tokens. It is perfectly possible for multiple nodes to align to the same token. It is not possible, within our framework, to represent a single node being sourced from multiple tokens.

There have been two previous attempts at producing automatic AMR alignments. The first was published as a necessary component of JAMR, (Flanigan et al., 2014), and used a rule-based approach to perform alignments, which worked well on the small sample of 100 hand-labeled sentences used to develop the system. The second published approach, **TODO:** cite short paper, rendered AMR graphs as text, and then used traditional alignment techniques from machine translation to align tokens in the source text and nodes in the AMR graphs. This approach works reasonably well, but fails to take advantage of the inherently graphical structure of AMR, and regularities within that structure like named entities and quantity values.

Our decomposition of the AMR node generation process into a set of actions provides an interesting way to align unaligned AMR graphs. We would like an alignment of AMR nodes to the source tokens such that we maximize the "informativeness" of the actions that we use to generate the AMR nodes from the source text.

We can define the "informativeness" of a given action by the probability of generating the correct nodes given the correct sequence label. The only label with a probability of correct generation that is less than 1 (i.e. is not and immediate guaranteed win) is **DICT**, which looks up the token in a dictionary, and on our dev set less than 70% are correctly generated from a **DICT**.

That suggests a relatively simple heuristic for producing good alignments: minimize the number of **DICT** sequence labels implied by a given alignment A. We would also like to constrain nodes that are not adjacent to one another to not align to the same token, except in certain cases where hallucinated AMR node structure suggests that a contiguous segment of 3 or more nodes is plausible.

TODO: Describe Boolean LP formulation and solution

6 Results

Our end to end results are reported by plugging the output of our subgraph generation system into the constrained-MST component of JAMR, which is able to produce final AMR graphs when given a sequence of spans and their corresponding chunks. AMR parsing accuracy is measured with a metric called smatch [citation needed], which stands for "s(emantic) match". We trained and tested on the **LDC2013E117** dataset, for which the only published result is a smatch score of **0.58** on the test set by JAMR (Flanigan et al., 2014). We report **0.64** on the same dataset, by substituting our subgraph generation system.

7 Future Work

7.1 Semantically equivalent POS normalization

The benefit of this approach could be increased by having a very strong stemmer tuned to AMR parsing, which currently doesn't exist.

7.2 Etymological approach to generation

There is an opportunity to create and test etymological-semantic approaches to parsing words like 'sailor' that would benefit AMR parsing domain generalization tremendously.

8 Appendix

NER++ Features
Input token
Input token word embedding
Left token
Right token
Left bigram
Right bigram
POS
Left POS
Right POS
Left POS bigram
Right POS bigram
Token's dependency parent token
Token's dependency parent POS
Token's dependency parent arc name
Bag of outgoing dependency arcs
Number of outgoing dependency arcs
Number of outgoing dependency arcs (indicator)
Max JaroWinker to any lemma in PropBank
Closest (JaroWinkler) in PropBank
Token NER
Left NER bigram
Right NER bigram
Right NER bigram
Indicator for if token is a recognized AMR NER type
Indicator for if token is capitalized
Parent arc is prep_* or appos, and parent has NER tag
Indicator for token is pronoun
Indicator for token is part of a coref chain
Indicator for token pronoun and part of a coref chain

Table 2: The features for the NER++ max-ent classifiers.

Acknowledgments

The acknowledgments should go immediately before the references. Do not number the acknowledgments section. Do not include this section when submitting your paper for review.

References

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, Noah A. Smith 2014. *ACL 14*, volume 1.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. *Proc. of the Linguistic Annotation Workshop and Iteroperability with Discourse*, volume 1.

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503–512.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.