Thesis Proposal:
Mapping Natural Language to and from the Abstract Meaning Representation

Jeffrey Flanigan

April 1, 2016

Abstract

A key task in intelligent language processing is obtaining semantic representations that abstract away from surface lexical and syntactic decisions. The Abstract Meaning Representation (AMR) is one such representation, which represents the meaning of a sentence as labeled nodes in a graph (concepts) and labeled, directed edges between them (relations). A traditional problem of semantic representations is producing them from natural language as well as producing natural language from them, or in other words, mapping into and out of the representation. In this thesis proposal, I discuss methods and algorithms for mapping into and out of AMR, as well as an application of these techniques to the multi-lingual setting.

Contents

1 Overview

This thesis is about producing meaning representations (MRs) from natural language and producing natural language from MRs, or in other words, mapping in both directions between natural language and MRs. The major contributions are algorithms for learning the mapping between text and MR using supervised structured prediction methods. The first two parts of this thesis are about parsing English into MR and generating English from MR, and the final part is about cross-lingual parsing and its application to machine translation.

The representation I use is the Abstract Meaning Representation (AMR) [Banarescu et al., 2013, Dorr et al., 1998], which has been designed with the idea of using it as an intermediate representation in machine translation. AMR represents the meaning of a sentence as label nodes in a graph (concepts), and edges between them (relations). It uses an inventory of concepts from English and PropBank [Palmer et al., 2005], and captures “who-is-doing-what-to-whom" in a propositional style logic, abstracting away from variations in surface syntax. AMR is not an interlingua, because it uses an inventory of concepts based on a natural language lexicon. A key recent development in AMR is that is has been used in a large annotation effort [Banarescu et al., 2013]. So far over 39,000 sentences have been annotated [Knight et al., 2016].

With annotated data, it becomes conceivable to train supervised algorithms to map in both directions between natural language text and AMR. In the first part of the proposal, I demonstrate that it is indeed possible to learn to map English text into AMR using this corpus (**semantic parsing from English to AMR**), and develop algorithms for doing this. In the second part of the proposal, I demonstrate that it is also possible to learn to map from AMR to English text (**generation of English from AMR**).[1] In the third section, I propose to map foreign language text to English AMR (**cross-lingual semantic parsing**).

While I develop algorithms and techniques for AMR, they are expected to be applicable to any model of semantics that represents the meaning of a sentence as a graph. While syntax is usually captured with

[1]With some underspecification in the generated English because AMR does not express all the semantics English does.

tree structures, it is often argued that propositional semantics is better represented with graph structures [Banarescu et al., 2013, Melcuk, 1988]. Desire to move from syntactic analysis to deeper semantic analysis has stimulated research into new algorithms that can handle these graph structures. This thesis contributes to this line of research.

It is possible that AMR or some other MR will significantly advance the state of the art in one or more of NLP tasks. Because AMR captures relations between concepts but abstracts away from surface syntax, it may be used as an intermediate representation in a variety of tasks such as MT, summarization, and question answering (QA). In a preliminary study, my co-authors and I have used AMR as an intermediate representation for summarization [Liu et al., 2015], and based on the original motivation and origins of AMR, future work on AMR applied to MT is expected. In fact, recently AMR has been used to considerably advance the state of the art in a set of QA tasks [Mitra and Baral, 2016]. Further research is necessary to determine if AMR or some other MR will prove to be widely useful in these tasks. The ability to learn mappings from natural language to MRs from annotated corpora is an important step for building systems that use intermediate MRs.

The rest of this section is an overview of the thesis proposal, followed by the thesis statement and contributions.

1.1 Parsing (completed work)

Parsing English into AMR is taking an input English sentence and producing an AMR graph as output. Unlike previous approaches to building parsers for meaning representations, in this work a broad coverage parser is learned from example sentences paired with their meaning representations.

The approach I take is a two-stage approach that first identifies the concepts in a sentence with a sequence labeling algorithm, and then adds relations between them to produce a well-formed AMR graph. I will also include recent progress in concept identification and relation identification. This is completed work. I will also include a possible extension applying the cross-lingual global decoder to the mono-lingual case.

1.2 Generation (completed work)

Generation from AMR is the production of a sentence from an input AMR graph. This is important for downstream applications such as summarization and machine translation, and success in this task increases AMR's usefulness as an intermediate representation.

My approach to generating English sentences from AMR is to first compute one or more spanning trees of the AMR graph, and then use a tree transducer to convert the tree into an English sentence. The tree transducer rules are learned from the AMR corpus, and there are also synthetic rules that are created on the fly. The system is trained discriminatively, with various features which include a language model. This is completed work.

1.3 Cross-lingual parsing (proposed work)

Cross-lingual AMR parsing is the production of English AMR graphs from foreign language text. This will be useful for downstream tasks such as machine translation and cross-lingual information extraction. I will describe my approach to cross-lingual semantic parsing, as well as two baselines.

My approach to cross-lingual parsing is to parse the English side of parallel bi-text with the English AMR parser, and then project the AMR graphs to the source side using automatic word alignments to create training data to train a source language to English AMR parser. To handle the extra ambiguity in the cross-lingual parsing, an new decoder is developed. Unlike the mono-lingual parser, the new decoder identifies concepts and relations jointly, can handle arbitrary features, and uses randomized greedy inference for approximate decoding. This is proposed work.

1.4 Thesis statement

> A key task in intelligent language processing is obtaining a representation of natural language expressions that abstracts away from surface lexical and syntactic decisions, in favor of an abstract semantic representation. I show that we can map into and out of the Abstract Meaning Representation (AMR) formalism using supervised structured prediction techniques based on graphs.

1.5 Contributions of this thesis

The main contributions of this thesis are:

- A two-stage approach to semantic parsing for AMR, where concepts are first identified using a sequence labeling algorithm, and relations are then added using a graph algorithm.
- An approximate inference algorithm for finding the maximum weight, spanning connected subgraph of a graph with linear constraints and weights on the edges.
- A two-stage approach to generation from AMR, where the graph is first converted to a tree, and then the tree is transduced into a string.
- A rule-based approach to automatic alignment of AMR graphs to the training data sentences, which is used to train the semantic parsers and generators.
- A joint approach to AMR parsing using a randomized greedy inference graph algorithm (see contribution below), which identifies concepts and relations together under a global scoring function.
- An approximate inference algorithm for finding the maximum connected subgraph of a graph with linear constraints and arbitrary scoring function.
- An approach to inducing cross-lingual AMR parsers from bi-text by automatically parsing the target and projecting the annotations to the source.
- A new loss function which generalizes SVM loss for structured prediction when the training data contains unreachable training examples.

2 Parsing

Parsing English into AMR is taking an input English sentence and producing an AMR graph as output. This is a necessary step for any system wishing to use AMRs for sentences not found in the AMR training corpus.

In parsing to AMR, both concepts and relations must be predicted. I solve this problem with a pipelined approach that first predicts concepts (§2.1) and then relations (§2.2). The pipeline for an example sentence is shown in Figure 1.

This section is mostly completed work, and largely follows Flanigan et al. [2014]. One extension, which is also completed work, is in parameter learning (§2.3) where I introduce a new loss function for learning called infinite ramp. This loss function is used for boosting concept fragment recall and obtaining re-entrancies in the AMR graphs.

Proposed work (§2.6) applies the joint parser developed for cross-lingual parsing (§4) to the monolingual case, with features developed for this setting.

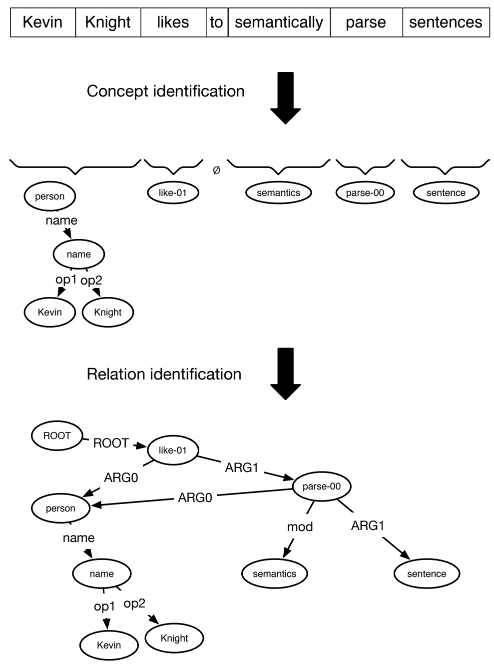


Figure 1: Stages in the parsing pipeline: concept identification followed by relation identification.

2.1 Concept Identification

The concept identification stage produces a set of concepts from the input sentence $\boldsymbol{w} = \langle w_1 \dots w_n \rangle$. It is treated as a sequence labeling problem, and each word or span of words is labeled with a concept fragment from a concept fragment set F, or $\emptyset$ for words the evoke no concepts.[2] Formally, concept identification is (i) a segmentation of $\boldsymbol{w}$ into contiguous spans represented by boundaries $\boldsymbol{b}$, giving spans $\langle \boldsymbol{w}_{b_0:b_1}, \boldsymbol{w}_{b_1:b_2}, \dots \boldsymbol{w}_{b_{k-1}:b_k} \rangle$, with $b_0 = 0$ and $b_k = n$, and (ii) an assignment of each phrase $\boldsymbol{w}_{b_{i-1}:b_i}$ to a concept graph fragment $c_i \in F \cup \{\emptyset\}$.

To perform concept identification, the decoder finds the maximum scoring sequence of spans $\boldsymbol{b}$ and sequence of concept graph fragments $\boldsymbol{c}$, both of arbitrary length k, according to a locally decomposed, linearly parameterized function:

$$score(\boldsymbol{b}, \boldsymbol{c}; \boldsymbol{\theta}) = \sum_{i=1}^{k} \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{w}_{b_{i-1}:b_i}, b_{i-1}, b_i, c_i) \tag{1}$$

where $\mathbf{f}$ is a feature vector representation of a span and one of its concept graph fragments in context. The features are:

[2] I call a set of concepts with relations between them a concept fragment. About 80% of invoked concept fragments are single concepts.

- **Fragment given words**: Relative frequency estimates of the probability of a concept graph fragment given the sequence of words in the span. This is calculated from the concept-word alignments in the training corpus.

- **Length** of the matching span (number of tokens).

- **NER**: 1 if the named entity tagger marked the span as an entity, 0 otherwise.

- **Bias**: 1 for any concept graph fragment from F and 0 for $\emptyset$.

The decoder uses a concept lexicon $clex$, which is a mapping $W^* \to 2^F$, to provide candidate graph fragments for sequences of words. (The construction of F and $clex$ is discussed below.)

The highest-scoring $\boldsymbol{b}$ and $\boldsymbol{c}$ is found exactly in $O(n^2)$ time using dynamic programming. Let $S(i)$ denote the score of the best labeling of the first i words of the sentence, $\boldsymbol{w}_{0:i}$. $S(i)$ can be calculated using the recurrence:

$$
\begin{aligned}
&S(0) = 0 \\
&S(i) = \max_{\substack{j:0 \leq j < i, \\ c \in clex(\boldsymbol{w}_{j:i}) \cup \emptyset}} \left\{ S(j) + \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{w}_{j:i}, j, i, c) \right\}
\end{aligned}
$$

The best score is $S(n)$, and the best scoring concept labeling is recovered using back-pointers, as in typical implementations of the Viterbi algorithm.

The function $clex : W^* \to 2^F$ is implemented as follows. When $clex$ is called with a sequence of words, it looks up the sequence in a table that contains, for every word sequence that was labeled with a concept fragment in the training data, the set of concept fragments it was labeled with. $clex$ also has a set of rules for generating concept fragments for named entities and time expressions. It generates a concept fragment for any entity recognized by the named entity tagger, as well as for any word sequence matching a regular expression for a time expression. $clex$ returns the union of all these concept fragments.

2.2 Relation Identification

Relation identification connects the concept graph fragments together to form a connected graph $G = \langle V_G, E_G \rangle$ by adding labeled directed edges.

Let $D = \langle V_D, E_D \rangle$ be the graph that includes all the labeled vertices and labeled edges in concept graph fragments, as well as every possible labeled edge $u \xrightarrow{\ell} v$, for all $u, v \in V_D$ and every $\ell \in L_E$. The output graph G is the maximum scoring subgraph of D under an edge-factored linear model, that satisfies the following constraints:

1. **Preserving**: all graph fragments (including labels) from the concept identification phase are subgraphs of G.

2. **Simple**: for any two vertices u and $v \in V_G$, E_G includes at most one edge between u and v. This constraint forbids a small number of perfectly valid graphs, for example for sentences such as "John hurt himself"; however, a small number $< 1\%$ of training instances violate the constraint. I found in preliminary experiments that including the constraint increases overall performance.

3. **Connected**: G must be weakly connected (every vertex reachable from every other vertex, ignoring the direction of edges). This constraint follows from the formal definition of AMR and is never violated in the training data.

Name	Description
Label	For each $\ell \in L_E$, 1 if the edge has that label
Self edge	1 if the edge is between two nodes in the same fragment
Tail fragment root	1 if the edge's tail is the root of its graph fragment
Head fragment root	1 if the edge's head is the root of its graph fragment
Path	Dependency edge labels and parts of speech on the shortest syntactic path between any two words in the two spans
Distance	Number of tokens (plus one) between the two concepts' spans (zero if the same)
Distance indicators	A feature for each distance value, that is 1 if the spans are of that distance
Log distance	Logarithm of the distance feature plus one.
Bias	1 for any edge.

Table 1: Features used in relation identification. In addition to the features above, the following conjunctions are used (Tail and Head concepts are elements of L_V): Tail concept $\wedge$ Label, Head concept $\wedge$ Label, Path $\wedge$ Label, Path $\wedge$ Head concept, Path $\wedge$ Tail concept, Path $\wedge$ Head concept $\wedge$ Label, Path $\wedge$ Tail concept $\wedge$ Label, Path $\wedge$ Head word, Path $\wedge$ Tail word, Path $\wedge$ Head word $\wedge$ Label, Path $\wedge$ Tail word $\wedge$ Label, Distance $\wedge$ Label, Distance $\wedge$ Path, and Distance $\wedge$ Path $\wedge$ Label. To conjoin the distance feature with anything else, multiply by the distance.

4. **Deterministic**: For each node $u \in V_G$, and for each label $\ell \in L_E^*$, there is at most one outgoing edge in E_G from u with label ℓ. This constraint is linguistically motivated and prevents a predicate from having more than one core argument of each type.

The scoring function for G decomposes by edges, and has the following linear parameterization:

$$score(E_G; \boldsymbol{\psi}) = \sum_{e \in E_G} \boldsymbol{\psi}^\top \mathbf{g}(e) \tag{2}$$

The vectors $\mathbf{g}(e)$ are the edge features shown in Table 1.

The solution to Eq. 2 subject to the constraints is found with (i) an algorithm that ignores constraint 4 but respects the others (§2.2.1) and (ii) Lagrangian relaxation that enforces constraint 4 (§2.2.2).

2.2.1 Maximum Preserving, Simple, Spanning, Connected Subgraph Algorithm

The steps below find the maxiumum subgraph satisfying constraints 1-3 in $O(n^2)$ time.

1. (Initialization) Let $E^{(0)}$ be the union of the concept graph fragments' weighted, labeled, directed edges. Let V denote its set of vertices. Note that $\langle V, E^{(0)} \rangle$ is preserving (constraint 1), as is any graph that contains it. It is also simple (constraint 2), assuming each concept graph fragment is simple.

2. (Pre-processing) Form the edge set E by including just one edge from E_D between each pair of nodes:

   - For any edge $e = u \xrightarrow{\ell} v$ in $E^{(0)}$, include e in E, omitting all other edges between u and v.
   - For any two nodes u and v, include only the highest scoring edge between u and v.

   Note that any subgraph of $\langle V, E \rangle$ is simple (constraint 2).

3. (Core algorithm) Run Algorithm 1, MSCG, on $\langle V, E \rangle$ and $E^{(0)}$. This algorithm is a novel modification of the minimum spanning tree algorithm of [Kruskal, 1956].

input : weighted, connected graph $\langle V, E\rangle$ and set of edges $E^{(0)} \subseteq E$ to be preserved
output: maximum spanning, connected subgraph of $\langle V, E\rangle$ that preserves $E^{(0)}$

let $E^{(1)} = E^{(0)} \cup \{e \in E \mid \boldsymbol{\psi}^\top \mathbf{g}(e) > 0\}$;
create a priority queue Q containing $\{e \in E \mid \boldsymbol{\psi}^\top \mathbf{g}(e) \leq 0\}$ prioritized by scores;
$i = 1$;
while *Q nonempty and $\langle V, E^{(i)}\rangle$ is not yet spanning and connected* **do**
    $i = i + 1$;
    $E^{(i)} = E^{(i-1)}$;
    $e = \arg\max_{e' \in Q} \boldsymbol{\psi}^\top \mathbf{g}(e')$;
    remove e from Q;
    if *e connects two previously unconnected components of $\langle V, E^{(i)}\rangle$* **then**
        add e to $E^{(i)}$
    end
end
return $G = \langle V, E^{(i)}\rangle$;

Algorithm 1: MSCG algorithm.

2.2.2 Lagrangian Relaxation

If the graph returned by the algorithm in §2.2.1 satisfies constraint 4, nothing more needs to be done. Otherwise, Lagrangian relaxation (LR) is used to attempt to enforce this constraint. LR is an approximate algorithm, although one can check when the solution returned is exact, as discussed below.

I begin by encoding a graph $G = \langle V_G, E_G\rangle$ as a binary vector. For each edge e in the fully dense multigraph D, associate a binary variable $z_e = \mathbf{1}\{e \in E_G\}$, where $\mathbf{1}\{P\}$ is the indicator function, taking value 1 if the proposition P is true, 0 otherwise. The collection of z_e form a vector $\mathbf{z} \in \{0,1\}^{|E_D|}$.

Determinism constraints can be encoded as a set of linear inequalities. For example, the constraint that vertex u has no more than one outgoing ARG0 can be encoded with the inequality:

$$\sum_{v \in V} \mathbf{1}\{u \xrightarrow{\text{ARG0}} v \in E_G\} = \sum_{v \in V} z_{u \xrightarrow{\text{ARG0}} v} \leq 1.$$

All of the determinism constraints can collectively be encoded as one system of inequalities:

$$\mathbf{A}\mathbf{z} \leq \mathbf{b},$$

with each row $\mathbf{A}_i$ in A and its corresponding entry b_i in $\mathbf{b}$ together encoding one constraint.

The score of graph G (encoded as $\mathbf{z}$) can be written as the objective function $\boldsymbol{\phi}^\top \mathbf{z}$, where $\phi_e = \boldsymbol{\psi}^\top \mathbf{g}(e)$. To handle the constraint $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, introduce multipliers $\boldsymbol{\mu} \geq 0$ to get the Lagrangian relaxation of the objective function:

$$L_{\boldsymbol{\mu}}(\mathbf{z}) = \boldsymbol{\phi}^\top \mathbf{z} + \boldsymbol{\mu}^\top(\mathbf{b} - \mathbf{A}\mathbf{z}),$$

$$\mathbf{z}^*_{\boldsymbol{\mu}} = \arg\max_{\mathbf{z}} L_{\boldsymbol{\mu}}(\mathbf{z}).$$

And the solution to the dual problem:

$$\boldsymbol{\mu}^* = \arg\min_{\boldsymbol{\mu} \geq \mathbf{0}} L_{\boldsymbol{\mu}}(\mathbf{z}^*_{\boldsymbol{\mu}})$$

$L_{\boldsymbol{\mu}}(\mathbf{z})$ decomposes over edges, so

$$\begin{aligned}\mathbf{z}^*_{\boldsymbol{\mu}} &= \arg\max_{\mathbf{z}} \left(\boldsymbol{\phi}^\top \mathbf{z} + \boldsymbol{\mu}^\top(\mathbf{b} - \mathbf{A}\mathbf{z})\right) \\ &= \arg\max_{\mathbf{z}} \left(\boldsymbol{\phi}^\top \mathbf{z} - \boldsymbol{\mu}^\top \mathbf{A}\mathbf{z}\right) \\ &= \arg\max_{\mathbf{z}} \left((\boldsymbol{\phi} - \mathbf{A}^\top \boldsymbol{\mu})^\top \mathbf{z}\right).\end{aligned}$$

Thus for any $\boldsymbol{\mu}$, one can find $\mathbf{z}^*_\mu$ by assigning edges the new Lagrangian adjusted weights $\boldsymbol{\phi} - \mathbf{A}^\top \boldsymbol{\mu}$ and reapplying the algorithm described in §2.2.1. $\mathbf{z}^* = \mathbf{z}^*_{\boldsymbol{\mu}^*}$ is found by projected subgradient descent, by starting with $\boldsymbol{\mu} = \mathbf{0}$, and taking steps in the direction:

$$-\frac{\partial L_{\boldsymbol{\mu}}}{\partial \boldsymbol{\mu}}(\mathbf{z}^*_{\boldsymbol{\mu}}) = \mathbf{A}\mathbf{z}^*_{\boldsymbol{\mu}} - \mathbf{b}.$$

If any components of $\boldsymbol{\mu}$ are negative after taking a step, they are set to zero.

$L_{\boldsymbol{\mu}^*}(\mathbf{z}^*)$ is an upper bound on the optimal solution to the primal constrained problem, and is equal to it if and only if the constraints $A\mathbf{z}^* \leq \mathbf{b}$ are satisfied. If $L_{\boldsymbol{\mu}^*}(\mathbf{z}^*) = \boldsymbol{\phi}^\top \mathbf{z}^*$, then $\mathbf{z}^*$ is also the optimal solution to the original constrained problem. Otherwise, there exists a duality gap, and Lagrangian relaxation has failed. In that case, the subgraph encoded by $\mathbf{z}^*$ is still returned, even though it might violate one or more constraints.

The runtime of LR is worst case exponential.[3]

2.3 Parameter Learning

The model parameters $\boldsymbol{\theta}$ for concept ID and $\boldsymbol{\psi}$ for relation ID are learned separately using aligned training data. The training data for the concept identification stage consists of (X, Y) pairs:

- **Input:** X, a sentence annotated with named entities (person, organization, location, misciscellaneous) from the Illinois Named Entity Tagger [Ratinov and Roth, 2009], and part-of-speech tags and basic dependencies from the Stanford Parser [Klein and Manning, 2003, de Marneffe et al., 2006].
- **Output:** Y, the sentence labeled with concept subgraph fragments.[4]

The training data for the relation identification stage consists of (X, Y) pairs:

- **Input:** X, the sentence labeled with graph fragments, as well as named enties, POS tags, and basic dependencies as in concept identification.
- **Output:** Y, the sentence with a full AMR parse.[5]

The parameters for each stage are learned using AdaGrad [Duchi et al., 2011] with the following loss function:

$$\text{loss}_{\infty\ \text{Ramp}} = \sum_{i=1}^{n} -\lim_{\alpha\to\infty} \max_{y\in\text{Gen}(x_i)} \vec{w}\cdot\vec{f}(x_i, y) - \alpha \cdot cost(y_i, y) + \max_{y\in\text{Gen}(x_i)} \vec{w}\cdot\vec{f}(x_i, y) + cost(y_i, y) \quad (3)$$

[3] The dual problem, as a linear program, can be solved in polynomial time with the ellipsoid algorithm, although this does not solve the primal because there may be a duality gap.

[4] Alignments are used to induce the concept labeling for the sentences, so no annotation beyond the automatic alignments is necessary.

[5] Because the alignments are automatic, some concepts may not be aligned, so one cannot compute their features. I remove the unaligned concepts and their edges from the full AMR graph for training. Thus some graphs used for training may in fact be disconnected.

P	R	F_1
.78	.81	.80

Table 2: Concept identification performance on test.

Concepts	P	R	F_1
gold	.76	.66	.71
automatic	.68	.60	.64

Table 3: Parser performance on test.

This loss function is a new loss function which I call infinite ramp. It is a generalization of the SVM loss [Cortes and Vapnik, 1995, Taskar et al., 2003, Tsochantaridis et al., 2004] and latent SVM loss [Yu and Joachims, 2009] to the case where the gold standard (x_i, y_i) is not in the output search space $\mathcal{Y}_i$ of the decoder. It reduces to them with the appropriate $cost(y_i, y)$.

2.4 Evaluation

I use Smatch [Cai and Knight, 2013] to evaluate the final output of the parser. This metric was developed for scoring the similarity between AMR graphs for inter-annotator agreement and for parser evaluation. Smatch counts corpus level precision, recall, and F_1 of concepts and relations together between two corpora of AMR graphs. I also report precision, recall, and F_1 for concept fragments, as well as Smatch scores using gold concepts to evaluate each stage of the pipeline separately. Evaluation is on the newest public release of the AMR dataset (LDC2014T12) with the recommended train/dev./test split.

2.5 Experiments

I report experimental results for concept ID (Table 2), and the full pipeline (Table 3) with automatic concepts as well as gold concepts.

2.6 Status and proposed work

This is completed work. As proposed work, I will apply the parser developed in §4 for cross-lingual parsing to mono-lingual parsing, with features designed for the mono-lingual task.

3 Generation

Generation of English from AMR is the production of an English sentence from an input AMR graph. This is important for downstream applications that wish to use AMR as an intermediate representation, such as MT [Jones et al., 2012] and summarization [Liu et al., 2015].

Generation of English from AMR graphs is accomplished as follows: the input graph is first converted to a tree (§3.2), which is input into a tree-to-string transducer (§3.2), and the output of the transducer is a weighted intersection with a language model; the output English sentence is the highest scoring sentence according to a feature-rich discriminatively trained linear model. The rules of the tree-to-string transducer are learned from the training data, and are of two types: rules extracted from the training data AMR graphs (**basic rules**, §3.3), and rules created after seeing the input AMR graph using a special rule creation model (**synthetic rules**, §3.4) — the latter help overcome sparsity.

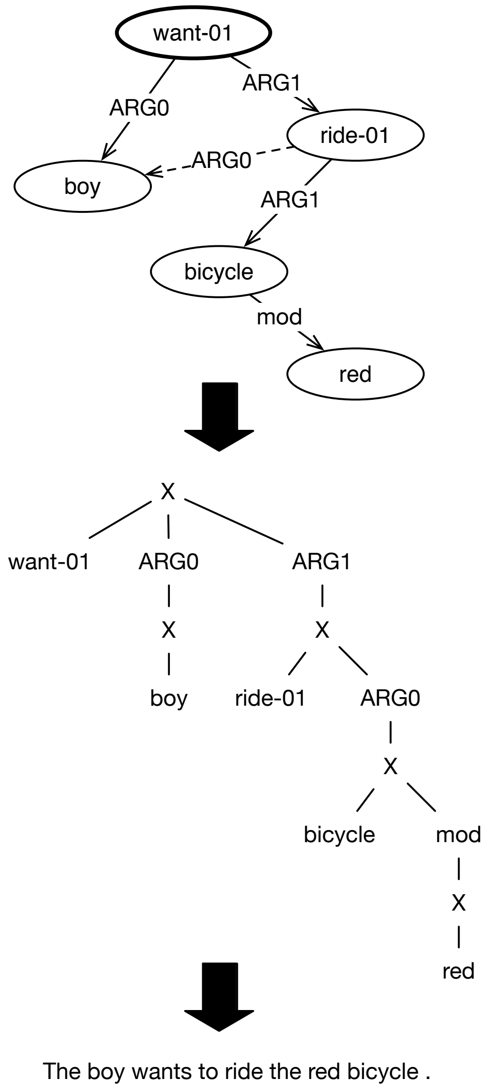


Figure 2: The generation pipeline. An AMR graph (top), with a deleted re-entrancy (dashed), is converted into a transducer input (middle), which is transduced to a string using a tree-to-string transducer (bottom).

3.1 Notation and definitions

AMR graphs are directed, weakly connected graphs with node labels from the set of **concepts** L_N and edge labels from the set of **relations** L_E. A **fragment** is a subgraph of a graph.

AMR graphs are transformed to eliminate cycles (details in §3.2); I refer to the resulting tree as a "transducer input." For a node n with label $\mathcal{C}$ and outgoing edges $n \xrightarrow{L_1} n_1, \ldots, n \xrightarrow{L_m} n_m$ sorted lexicographically by L_i, the transducer input of the tree rooted at n is denoted:[6]

$$(X\ \mathcal{C}\ (L_1\ T_1)\ldots(L_m\ T_m)) \tag{4}$$

where each T_i is the transducer input of the tree rooted at n_i. See Fig. 2 for an example. A LISP-like textual formatting of the transducer input in Fig. 2 is:

$$(X\ \textit{want-01}\ (\textit{ARG0}\ (X\ \textit{boy}))\ (\textit{ARG1}\ (X\ \textit{ride-01}\ (\textit{ARG0}\ (X\ \textit{bicycle}\ (\textit{mod}\ (X\ \textit{red})))))))$$

To ease notation, I use the function $sort[]$ to lexicographically sort edge labels in a transducer input. Using this function, an equivalent way of representing the transducer input in Eq. 4, if the L_i are unsorted, is:

$$(\mathrm{X}\ \mathcal{C}\ sort[(L_1\ T_1)\ldots(L_m\ T_m)])$$

The transducer input is converted into a word sequence using a tree-to-string transducer. The tree transducer formalism used is one-state extended linear, non-deleting tree-to-string (**1-xRLNs**) transducers [Huang et al., 2006, Graehl and Knight, 2004].[7]

Definition 1. *(From Huang et al., 2006.) A **1-xRLNs transducer** is a tuple $(N, \Sigma, W, \mathcal{R})$ where Σ is the input alphabet (concept labels), N is the set of nonterminals (relation labels and X), W is the output alphabet (words), and $\mathcal{R}$ is the set of rules. A rule in $\mathcal{R}$ is a tuple (t, s, ϕ) where:*

1. *t is the LHS tree, whose internal nodes are labeled by nonterminal symbols, and whose frontier nodes are labeled terminals from Σ or variables from a set $\mathcal{X} = \{X_1, X_2, \ldots\}$;*
2. *$s \in (\mathcal{X} \cup W)^*$ is the RHS string;*
3. *ϕ is a mapping from $\mathcal{X}$ to nonterminals N.*

A rule is a **purely lexical rule** if it has no variables.

As an example, the tree-to-string transducer rules which produce the output sentence from the transducer input in Fig. 2 are:

$$\begin{gathered}(X\ \textit{want-01}\ (\textit{ARG0}\ X_1)\ (\textit{ARG1}\ X_2)) \rightarrow \text{The } X_1 \text{ wants to } X_2\ .\\ (X\ \textit{ride-01}\ (\textit{ARG1}\ X_1)) \rightarrow \text{ride the } X_1\\ (X\ \textit{bicycle}\ (\textit{mod}\ X_1)) \rightarrow X_1 \text{ bicycle}\\ (X\ \textit{red}) \rightarrow \text{red}\end{gathered} \tag{5}$$

The output string of the transducer is the target projection of the derivation, which are defined as follows:

Definition 2. *(From Huang et al., 2006.) A **derivation** d, its **source and target projections**, denoted $\mathcal{S}(d)$ and $\mathcal{E}(d)$ respectively, are recursively defined as follows:*

1. *If $r = (t, s, \phi)$ is a purely lexical rule, then $d = r$ is a derivation, where $\mathcal{S}(d) = t$ and $\mathcal{E}(d) = s$;*
2. *If $r = (t, s, \phi)$ is a rule, and d_i is a (sub)-derivation with the root symbol of its source projection matching the corresponding substition node in r, i.e., root$(\mathcal{S}(d_i)) = \phi(x_i)$, then $d = r(d_1, \ldots, d_m)$ is also a derivation, where $\mathcal{S}(d) = [x_i \mapsto \mathcal{S}(d_i)]t$ and $\mathcal{E}(d) = [x_i \mapsto \mathcal{E}(d_i)]s$.*

[6] If there are duplicate child edge labels, then the conversion process is ambiguous and any of the conversions can be used. The ordering ambiguity will be handled later in the tree-transducer rules.

[7] Multiple states would be useful for modeling dependencies in the output, but I do not use them here.

The notation $[x_i \mapsto y_i]$ is shorthand for the result of substituting y_i for each x_i in t, where x_i ranges over all variables in t. The set of all derivations of a target string e with a transducer T is denoted

$$\mathcal{D}(e, T) = \{d \mid \mathcal{E}(d) = e\}$$

where d is a derivation in T.

I use a shorthand notation for the transducer rules that will be useful when discussing rule extraction and synthetic rules. Let f_i be a transducer input, and let $A_1, \ldots A_n \in L_E$. The transducer input has the form

$$f_i = (X\ \mathcal{C}\ (L_1\ T_1) \ldots (L_m\ T_m))$$

where $L_i \in L_E$ and $T_1, \ldots, T_m$ are transducer inputs.[8] Let $A_1, \ldots A_n \in L_E$. I use

$$(f_i, A_1, \ldots, A_n) \rightarrow r \tag{6}$$

as shorthand for the rule:

$$(X\ \mathcal{C}\ sort[(L_1\ T_1)\ \ldots\ (L_m\ T_m)(A_1\ X_1)\ \ldots\ (A_n\ X_n)]) \rightarrow r$$

In (6), argument slots with relation labels A_i have been added as children to the root node of the transducer input f_i.

For example, the shorthand for the transducer rules in (5) is:

$$\begin{gathered}((X\ \textit{want-01}), \textit{ARG0}, \textit{ARG1}) \rightarrow \text{The } X_1 \text{ wants to } X_2\ .\\ ((X\ \textit{ride-01}), \textit{ARG1}) \rightarrow \text{ride the } X_1\\ ((X\ \textit{bicycle}), \textit{mod}) \rightarrow X_1 \text{ bicycle}\\ ((X\ \textit{red})) \rightarrow \text{red}\end{gathered} \tag{7}$$

3.2 Generation using a tree-to-string transducer

To generate a sentence e from an input AMR graph G, a spanning tree G' of G is computed, then transformed into a string using a tree-to-string transducer.

Spanning tree. The choice of the graph G's spanning tree G' could have a big effect on the output, since the transducer's output will always be a projective reordering of the tree's leaves. Our spanning tree results from a breadth-first-search traversal, visiting child nodes in lexicographic order of the relation label (inverse relations are visited last). The edges traversed are included in the tree. This simple heuristic is a baseline which can potentially be improved in future work.

Decoding. Let $T = (N, \Sigma, W, \mathcal{R})$ be a tree-to-string transducer. The output sentence is the highest scoring transduction of G':

$$e = \mathcal{E}\left(\operatorname*{arg\,max}_{d \in \mathcal{D}(G', T)} score(d; \boldsymbol{\theta})\right) \tag{8}$$

Eq. 8 is solved approximately using the cdec decoder for machine translation [Dyer et al., 2010]. The score of the transduction is a linear function (with coefficients $\boldsymbol{\theta}$) of a vector of features including the output

[8] If f_i is just a single concept with no children, then $m = 0$ and $f_i = (X\ \mathcal{C})$.

Name	Description
Rule	1 for every rule
Basic	1 for basic rules, else 0
Synthetic	1 for synthetic rules, else 0
Abstract	1 for abstract rules, else 0
Handwritten	1 for handwritten rules, else 0
Rule given concept	log(number of times rule extracted / number of times concept observed in training data) (only for basic rules, 0 otherwise)
Rule given concept w/o sense	same as above, but with sense tags for concepts removed
Synthetic score	model score for the synthetic rule (only for synthetic rules, 0 otherwise)
Word count	number of words in the rule
Stop word count	number of words not in a stop word list
Bad stop word	number of words in a list of meaning changing stop words, such as "all, can, could, only, so, too, until, very"
Negation word	number of words in "no, not, n't"

Table 4: Rule features in the transducer. There is also an indicator feature for every handwritten rule.

sequence's language model log-probability and features associated with the rules in the derivation (denoted $\mathbf{f}$; Table 4):

$$score(d;\boldsymbol{\theta}) = \theta_{LM}\log(p_{LM}(\mathcal{E}(d))) + \sum_{r\in d}\boldsymbol{\theta}^\top\mathbf{f}(r)$$

The feature weights are trained on a development dataset using MERT [Och, 2003].

3.3 Basic rules

The basic rules, denoted $\mathcal{R}_B$, are extracted from the training AMR data using an algorithm similar to extracting tree transducers from tree-string aligned parallel corpora [Galley et al., 2004]. Informally, the rules are extracted from a sentence $\boldsymbol{w} = \langle w_1, \ldots, w_n\rangle$ with AMR graph G as follows.

1. The AMR graph and the sentence are aligned; I use the JAMR aligner from Flanigan et al. [2014], which aligns fragments of the graph to spans of words.
2. G is replaced by its spanning tree by deleting relation that use a variable in the AMR annotation.
3. In the tree, for each node i, I keep track of the word indices $b(i)$ and $e(i)$ in the original sentence that trap all of i's descendants. (This is calculated using a simple bottom-up propagation from the leaves to the root.)
4. For each aligned fragment i, a rule is extracted by taking the subsequence $\langle w_{b(i)} \ldots w_{e(i)}\rangle$ and "punching out" the spans of the child nodes (and their descendants) and replacing them with argument slots.

See Fig. 3 for examples.

More formally, assume the nodes in G are numbered $1, \ldots, N$ and the fragments are numbered $1, \ldots, F$. A node is considered aligned if it belongs to an aligned frament. Let the span of an aligned node i be denoted by endpoints a_i and a'_i; for unaligned nodes, $a_i = \infty$ and $a'_i = -\infty$ (depicted with superscripts in Fig. 3). The node alignments are propagated by defining $b(\cdot)$ and $e(\cdot)$ recursively, bottom up:

$$b(i) = \min_{j\in\{i\}\cup\text{children}(i)} a_j$$

$$e(i) = \max_{j \in \{i\} \cup \text{children}(i)} a'_j$$

Also define functions $\tilde{b}$ and $\tilde{e}$, from fragment indices to integers, as:

$$\tilde{b}(i) = b(\text{root}(i))$$

$$\tilde{e}(i) = e(\text{root}(i))$$

For fragment i, let $C_i = \text{children}(\text{root}(i)) - \text{nodes}(i)$, which is the children of the fragment's root concept that are not included in the fragment. Let f_i be the transducer input for fragment i.[9] If C_i is empty, then the rule extracted for fragment i is:

$$r_i : (f_i) \rightarrow w_{\tilde{b}(i):\tilde{e}(i)} \tag{9}$$

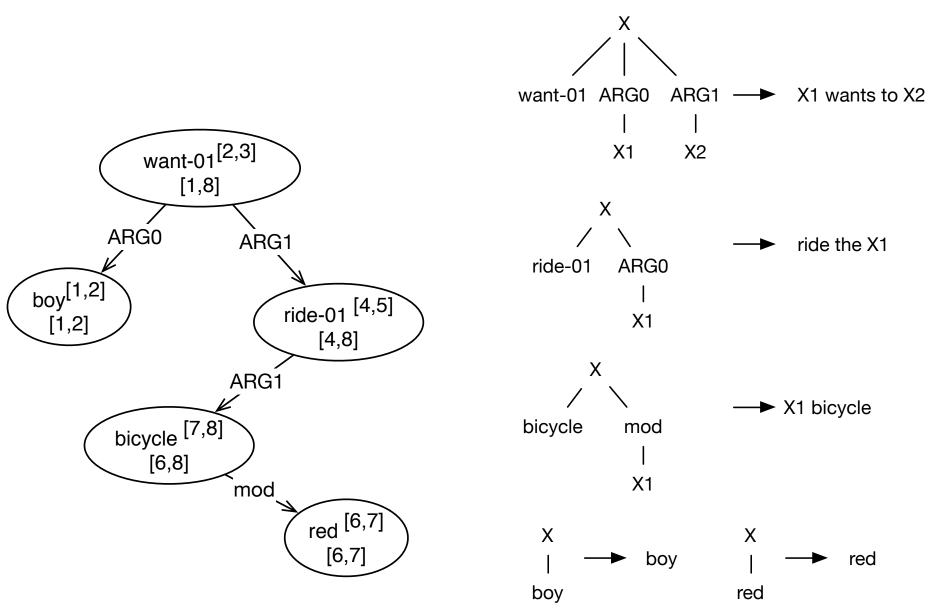
Otherwise, let $m = |C_i|$, and denote the edge labels from $\text{root}(i)$ to elements of C_i as $A_1(i) \ldots A_m(i)$. For $j \in \{1, \ldots, m\}$, let k_j select the elements c_{k_j} of C_i in ascending order of $b(k_j)$. Then the rule extracted for fragment i is:

$$r_i : (f_i, A_{k_1}(i), \ldots A_{k_m}(i)) \rightarrow w_{\tilde{b}(i):\tilde{b}(k_1)}\ X_1\ w_{\tilde{e}(k_1):\tilde{b}(k_2)}\ X_2 \ldots \ldots w_{\tilde{e}(k_{m-1}):\tilde{b}(k_m)} X_m\ w_{\tilde{e}(k_m):\tilde{e}(i)} \tag{10}$$

A rule is only extracted if the fragment i is aligned and the child spans do not overlap. Fig. 3 gives an example of a tree annotated with alignments, b and e, and the extracted rules.

$_0$ The $_1$ ((boy) $_2$ wants $_3$ to $_4$ (ride $_5$ the $_6$ ((red) $_7$ bicycle))) $_8$

(a) Sentence annotated with indexes, and bracketed according to $b(i)$ and $e(i)$ from the graph in (b).



(b) Tree annotated with a_i, a'_i (superscripts) and $b(i)$, $e(i)$ (subscripts).

(c) Extracted rules.

Figure 3: Example rule extraction from an AMR-annotated sentence.

[9] I.e., the nodes in fragment i, with the edges between them from T, represented as a transducer input.

3.4 Synthetic rules

The synthetic rules, denoted $\mathcal{R}_S(G)$, are created to generalize the basic rules and overcome data sparseness resulting from our relatively small training dataset. Our synthetic rule model considers an AMR graph G and generates a set of rules for each node in G. As for basic rules, a synthetic rule's LHS is a transducer input f with argument slots $A_1 \dots A_m$. For each node, one or more LHS are created (we will discuss this further below), and for each LHS, a set of k-best synthetic rules are produced. The simplest case of a LHS is just a concept and argument slots corresponding to each of its children.

For a given LHS, the synthetic rule model creates a RHS by concatenating together a string in W^* (called a **concept realization** and corresponding to the concept fragment) with strings in $W^*\mathcal{X}W^*$ (called an **argument realization** and corresponding to the argument slots). See Fig. 4 top for a synthetic rule with concept and argument realizations highlighted.

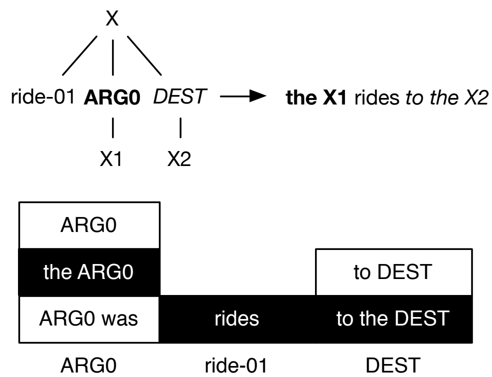


Figure 4: Synthetic rule generation for the rule shown at top. In the rule RHS, the realization for ARG0 is bold, and italic for DEST, and normal typeface for ride-01. For a fixed permutation of the concept and arguments, choosing the argument realizations is a sequence labeling problem (bottom). The highlighted sequence corresponds to the rule at top.

Synthetic rules have the form:

$$r : (f, A_1, \dots A_m) \to \mathbf{l}_{k_1} X_{k_1} \mathbf{r}_{k_1} \dots \mathbf{l}_{\mathbf{k_c}} X_{k_c} \mathbf{r}_{k_c}\ \mathbf{c} \mathbf{l}_{k_{c+1}} X_{k_{c+1}} \mathbf{r}_{k_{c+1}} \dots \mathbf{l}_{k_m} X_{k_m} \mathbf{r}_{k_m} \tag{11}$$

where:

- f is a transducer input.
- Each $A_i \in L_E$.
- $\langle k_1, \dots, k_m \rangle$ is a permutation of $\langle 1, \dots, m \rangle$
- $\mathbf{c} \in W^*$ is the **realization** of transducer input f.
- Each $\mathbf{l}_i, \mathbf{r}_i \in W^*$ and $X_i \in \mathcal{X}$. Let $R_i = \langle \mathbf{l}_i, \mathbf{r}_i \rangle$ denote the **realization of argument** i.
- $c \in [0, m]$ is the position of $\mathbf{c}$ among the realizations of the arguments.

Let $\mathcal{F}$ be the set of all possible transducer inputs. Synthetic rules make use of three lookup tables to provide candidate realizations for concepts and arguments: a table for concept realizations $lex : \mathcal{F} \to 2^{W^*}$, a table for argument realizations when the argument is on the left $\mathit{left}_{lex} : \mathcal{F} \times L_E \to 2^{W^*}$, and a table for argument realizations when the argument is on the right $\mathit{right}_{lex} : \mathcal{F} \times L_E \to 2^{W^*}$. These tables are constructed during basic rule extraction, the details of which are discussed below.

Synthetic rules are selected using a linear model with features $\mathbf{g}$ and coefficients $\boldsymbol{\phi}$, which scores each RHS for a given LHS. For LHS = $(f, A_1, \ldots A_m)$, the RHS is specified completely by $\mathbf{c}, c, R_1, \ldots, R_m$ and a permutation k_i. For each node in G, and for each transducer input f in the domain of lex that matches the node, a LHS is created, and a set of K synthetic rules is produced for each $\mathbf{c} \in lex(f)$. The rules produced are the K-best solutions to:

$$\begin{aligned}\operatorname*{arg\,max}_{c,k_1\ldots k_m,R_1,\ldots,R_m} \Big(&\sum_{i=1}^{c} \boldsymbol{\psi}^\top \mathbf{g}(R_{k_i}, A_{k_i}, \mathbf{c}, i, c) + \boldsymbol{\psi}^\top \mathbf{g}(\langle \epsilon, \epsilon \rangle, *, \mathbf{c}, c+1, c) \\ &+ \sum_{i=c+1}^{m} \boldsymbol{\psi}^\top \mathbf{g}(R_{k_i}, A_{k_i}, \mathbf{c}, i+1, c) \Big) \end{aligned} \tag{12}$$

where the max is over $c \in 0 \ldots m$, $k_1, \ldots, k_m$ is any permutation of $1, \ldots, m$, and $R_i \in \mathit{left}_{lex}(A_i)$ for $i < c$ and $R_i \in \mathit{right}_{lex}(A_i)$ for $i > c$. $*$ is used to denote the concept position. ϵ is the empty string.

The best solution to Eq. 12 is found by brute force search over concept position $c \in [0, m+1]$ and the permutation $k_1, \ldots, k_m$. With fixed concept position and permutation, each R_i for the $\arg\max$ is found independently. To obtain the K-best solutions, I use dynamic programming with a K-best semiring [Goodman, 1999] to keep track of the K best sequences for each concept position and permutation, and take the best K sequences over all values of c and k..

The synthetic rule model's parameters are estimated using basic rules extracted from the training data. Basic rules are put into the form of Eq. 11 by segmenting the RHS into the form

$$\mathbf{l}_1 X_1 \mathbf{r}_1 \ \ldots \ \mathbf{c} \ \ldots \ \mathbf{l}_m X_m \mathbf{r}_m \tag{13}$$

by choosing $\mathbf{c}, \mathbf{l}_i, \mathbf{r}_i \in W^*$ for $i \in 1, \ldots, m\}$.

An example segmentation is the rule RHS in Fig. 4. The segmented basic rules are also used to populate the tables for lex, left_{lex}, and right_{lex}.

Segmenting the RHS of the basic rules into the form of Eq. 13 is done as follows: $\mathbf{c}$ is the aligned span for f. For the argument realizations, arguments to the left of $\mathbf{c}$ pick up words to their right, and arguments to the right pick up words to their left. Specifically, for $i < c$ (R_i to the left of $\mathbf{c}$ but not next to $\mathbf{c}$), $\mathbf{l}_i$ is empty and $\mathbf{r}_i$ contains all words between a_i and a_{i+1}. For $i = c$ (R_i directly to the left of $\mathbf{c}$), $\mathbf{l}_i$ is empty and $\mathbf{r}_i$ contains all words between a_c and $\mathbf{c}$. For $i > c+1$, $\mathbf{l}_i$ contains all words between a_{i-1} and a_i, and for $i = c+1$, $\mathbf{l}_i$ contains all words between $\mathbf{c}$ and a_i.

The parameters $\boldsymbol{\psi}$ are trained using AdaGrad [Duchi et al., 2011] with the perceptron loss function [Rosenblatt, 1957, Collins, 2002] for 10 iterations over the basic rules. The features $\mathbf{g}$ are listed in Table 5.

3.5 Abstract rules

Like the synthetic rules, the abstract rules $\mathcal{R}_A(G)$ generalize the basic rules. However, abstract rules are much simpler generalizations which use part-of-speech (POS) tags to generalize. Abstract rules make use of a **POS abstract rule table**, which is a table listing every combination of the POS of the concept realization, the child arguments labels, and rule RHS with the concept realization removed and replaced with $*$. This table is populated from the basic rules extracted from the training corpus. An example entry in the table is:

$$(\mathrm{VBD}, \mathrm{ARG0}, \mathrm{DEST}) \to X_1 \ \langle * \rangle \ \text{to the } X_2$$

Feature name	**Value**
POS + A_i + "dist"	$\lvert c-i\rvert$
POS + A_i + side	1.0
POS + A_i + side + "dist"	$\lvert c-i\rvert$
POS + A_i + R_i + side	1.0
c + A_i + "dist"	$\lvert c-i\rvert$
c + A_i + side	1.0
c + A_i + side + "dist"	$\lvert c-i\rvert$
c + POS + A_i + side + "dist"	$\lvert c-i\rvert$

Table 5: Synthetic rule model features. POS is the most common part-of-speech tag sequence for **c**, "dist" is the string "dist", and side is "L" if $i < c$, "R" otherwise. $+$ denotes string concatenation.

Split	Sentences	Tokens
Train	10k	210k
Dev.	1.4k	29k
Test	1.4k	30k
MT09	204	5k

Table 6: Train/dev./test/MT09 split.

For the LHS $(f, A_1, \ldots A_m)$, an abstract rule is created for each member of $\mathbf{c} \in lex(f)$ and the most common POS tag p for **c** by looking up p, $A_1, \ldots A_m$ in the POS abstract rule table, finding the common RHS, and filling in the concept position with **c**. The set of all such rules is returned.

3.6 Handwritten Rules

There are handwritten rules for dates, conjunctions, multiple sentences, and the concept have-org-role-91. Pass-through rules for concepts are created by removing sense tags and quotes (for string literals).

3.7 Experiments

I evaluate on the LDC2014T12 dataset, following the recommended train/dev./test splits, except that MT09 data (204 sentences) is removed from the training data and use it as another test set. Statistics for this dataset and splits are given in Table 6. I use a 5-gram language model trained with KenLM [Heafield et al., 2013] on Gigaword (LDC2011T07), and use 100-best synthetic rules.

I evaluate with the BLEU scoring metric [Papineni et al., 2002] (Table 7), and report single reference BLEU for the LCD2014T12 test set, and four reference BLEU for the MT09 set. I also report ablation experiments for different sources of rules. When ablating handwritten rules, pass-through rules are not ablated.

The results are quite interesting. The full system achieves 22.1 BLEU on the test set, and 21.2 on MT09. Removing the synthetic rules drops the results to 9.1 BLEU on test and 7.8 on MT09. Removing the basic and abstract rules has little impact on the results. This may be because the synthetic rule model already contains much of the information in the basic and abstract rules. Removing the handwritten rules has a slightly larger effect, demonstrating the value of handwritten rules in this statistical system.

Rules	Test	MT09
Full	22.1	21.2
Full – basic	22.1	20.9
Full – synthetic	9.1	7.8
Full – abstract	22.0	21.2
Full – handwritten	21.9	20.5

Table 7: Uncased BLEU scores with various types of rules removed from the full system.

3.8 Status

This is completed work.

4 Application: Cross-lingual parsing

As an application, I consider mapping a language other than English into AMR. **Cross-lingual semantic parsing** is parsing from one language into a semantic representation based on a different target language. The most widely annotated version of AMR uses a concept inventory from the English language, but versions of AMR using concept inventories from other languages have been annotated at a small scale. I will call the language from which AMR takes its concept inventory and core argument relations the **base language**.

One approach to using AMR in machine translation is to use AMR as an intermediate semantic representation during translation. The source text is converted into an AMR graph, and then a target sentence is generated from the graph, possibly also using information from the source text. If the AMR base language is the target language, then this is cross-lingual semantic parsing followed by mono-lingual generation. If the AMR base language is the source language, then this is mono-lingual semantic parsing followed by cross-lingual generation. In both cases, the source text can be used as additional information during generation.

In this work, I focus on cross-lingual semantic parsing into English AMR. My approach is to use bilingual sentence-aligned parallel data, commonly used to build MT systems, to build a cross-lingual parser from the source language to the target language AMR. To build this parser, I develop a new approximate algorithm for joint concept and relation identification under a global scoring function, which uses hill-climbing with random restarts for inference. At its core, the decoder has an approximate algorithm for finding the maximum connected subgraph of a graph with linear constraints and a global scoring function. I will refer this optimization problem as **MCG** (Max Connected sub-Graph).

I expect joint concept and relation identification to be necessary for cross-lingual parsing. Preliminary experiments applying the monolingual decoder (§2) to cross-lingual parsing have indicated that a language model during concept identification is necessary. Concepts in different languages have a many-to-many correspondence, and the mono-lingual decoder has trouble disambiguating concepts and deciding when to drop redundant concepts or add extra concepts. It is an open question what the language model should be for cross-lingual parsing.

I argue that for graphs, features of conjunctions of node and edge labels in the predicted graph are the analogue of a language model for sequences.[10] For example, a 4-gram language model can score the words "went to the store" or "plane in the sky", which in AMR corresponds to two concepts with a relation:

[10]These features don't need to be indicator features, they can be based on relative frequency estimates calculated from target monolingual data.

(g / go-01 :ARG1 (s / store)) and (p / plane :location (s / sky)). A 4-gram language model can also score the words "I went to the" or "I saw a plane", which is a conjunction of two relations and two or three concepts: (g / go-01 :ARG0 (i / i) :ARG1) and (s / see-01 :ARG0 (i / i) :ARG1 (p / plane)). These observations motivate joint concept and relation identification with higher-order features for cross-lingual parsing.

An alternative approach for adding a language model to cross-lingual parsing is to linearize the concepts using some method and apply a language model for sequences. However, the ability of a joint decoder to reason locally in the target graph but non-locally in a linear sequence is a strength of the joint approach.

Although I only describe one approach to cross-lingual parsing, and is the one I will try first, it is not the only approach I will consider. For example, one backup approach is to apply the neural model for translation with attention [Bahdanau et al., 2014] to concept identification, then run the relation identification algorithm from §2.2.

The reason for choosing a global decoder with randomized greedy inference is that this inference procedure allows the most flexibility in features and has been successful in syntactic dependency parsing. Hill-climbing with random restarts has been used to solve NP-hard problems such as the graph matching problem in Smatch [Cai and Knight, 2013], and has been quite successful in syntactic dependency parsing [Zhang et al., 2014]. In particular, it is state of the art for joint word segmentation, POS tagging and dependency parsing in Chinese [Zhang et al., 2015]. Formally this task is identical to joint concept and relation identification for AMR parsing, except it predicts trees instead of graphs. Therefore we expect an extension to graphs to work well for AMR parsing.

An added benefit of using the randomized greedy inference algorithm that I have developed for cross-lingual parsing (which approximately solves MCG) is that the algorithm is very general and can in theory be applied to other problems. Not only can it be applied to mono-lingual parsing, but it could also be used in the generator (§3) for inference with global features and joint synthetic rules.[11] With a minor extension, the algorithm can be used for joint cross-lingual parsing and generation (i.e. joint full translation) with global features. However, I am not proposing to do full translation and leave that task to future work.

The hill-climbing method that I use is related to **enforced hill-climbing (EHC)** search [Hoffmann and Nebel, 2001], which was invented for planning algorithms. EHC is widely used in heuristic-search based planning systems, in particular the FastForward (FF) planner [Hoffmann and Nebel, 2001], one of the most successful symbolic planners in recent years. EHC is a local search algorithm that combines hill-climbing with breadth-first search (BFS) to escape local optima. EHC is similar to, but differs from, the hill-climbing algorithm I use in the following ways: 1) it uses BFS instead of DFS, 2) exhaustive search is used to escape local optima whereas here it is used only to escape invalid configurations (violated constraints), 3) I use random restarts to escape local optima instead of exhaustive search. It is encouraging that such a similar approach is successful in other areas of computer science with difficult search problems, and points to some potential cross-fertilization of the various approaches.

4.1 Method overview

To build the source language to English AMR parser, the English side of the parallel data is parsed with the English parser described in §2. The alignments between the spans of words in the English sentence and the fragments in the automatic AMR parse are retained from the mono-lingual concept identification stage (§2.1) and are projected to the source language text using automatic word alignments (§4.3). The source language text with the aligned automatic AMR parses is used to train the cross-lingual AMR parser,

[11] More precisely, the inference problem of the generator (with or without joint synthetic rule inference) can be written as a max connected subgraph problem with linear constraints.

which uses randomized greedy inference as an approximate decoding algorithm (§4.4), and is trained using a discriminative learning method.

At its core, the decoder uses an approximate algorithm to find the maximum connected subgraph of a graph with linear constraints on the nodes and edges and an arbitrary global scoring function (MCG, §4.5).

4.2 Baselines

There are two baselines to which I will compare: 1) phrase-based machine translation and followed by English AMR parsing, and 2) phrase-based machine translation and followed by English concept identification, but using full cross-lingual relation identification with the algorithm from §2.2.

4.3 Alignment projection and phrase-concept pair extraction

In this section I describe how alignments are obtained between the source sentences and the target AMR graphs in the training data, and how they are used to extract possibly gappy source phrases paired with AMR concept fragments.

The mono-lingual parser (§2) is run on the target and outputs not only an AMR graph but also alignments back to spans of words in the target. The alignments to spans in the target are projected to possibly gappy spans in the source using automatic word alignments. During this process, because of the many-to-many alignment between words, some of the concept fragments that make up the AMR graph are merged into one fragment. This is done so that gappy-phrase to concept fragment pairs can be extracted without breaking word alignments. For example, if two concepts are aligned to one source word, then these concepts are merged into one fragment that contains two concepts with possibly edges between them. The exact details of the alignment projection are discussed below.

Consider a sentence pair in the training data, with source sentence $\langle \boldsymbol{s}_1, \ldots \boldsymbol{s}_m \rangle$ of length m, target sentence $\langle \boldsymbol{t}_1, \ldots \boldsymbol{t}_m \rangle$ of length n, and alignment matrix $\boldsymbol{a}$, where $a_{ij} = 1$ if the $\boldsymbol{s}_i$ is aligned to $\boldsymbol{t}_j$, and zero otherwise. The target sentence is parsed with the parser described in §2, giving a set of boundaries $\boldsymbol{b}$ for the target side spans $\langle \boldsymbol{t}_{b_0:b_1}, \boldsymbol{t}_{b_1:b_2}, \ldots \boldsymbol{t}_{b_{k-1}:b_k} \rangle$. The ith target side phrase $\boldsymbol{t}_{b_{i-1}:b_i}$ is labeled with the concept graph fragment $c_i \in F \cup \{\emptyset\}$.

Spans in the target aligned to concept fragments are projected to gappy spans in the source aligned to possibly redefined concept fragments as follows. Let $\mathbf{c}$ be a the concept alignment matrix, where $c_{ij} = 1$ indicates target concept fragment j is aligned to the source word $\boldsymbol{s}_i$, and zero otherwise. It is computed as follows:

$$c_{ij} = \begin{cases} 1 & \text{if } \exists k \in [b_{j-1}, b_j] \text{ such that } a_{ik} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Viewing the words in the source and the individual concept fragments as nodes a bi-partite graph, we extract all connected components of this graph as source-concept fragment pairs. I will refer to the source-concept fragment pairs as **phrase-concept pairs**. If present, we keep edges present between the concept fragments in an extracted phrase-concept pair. Using the extracted phrase-concept pairs, we compute the usual relative frequency estimates for Pr(extracted_fragment | gappy_phrase) and Pr(gappy_phrase | extracted_fragment) for use as features in the decoder.

4.4 Joint decoder with global features

To obtain the best target AMR parse for an input source sentence, the decoder jointly finds 1) the best labeling of possibly gappy spans in the source labeled with concept fragments, and 2) edges between the fragments to make them connected, subject to well-formedness constraints.

Consider an input sentence. A phrase-concept pair extracted from the training data matches the sentence if the words in the phrase-concept pair are in the same order as the words it matches in the sentence, and has gaps only where the phrase-concept pair has gaps (although gaps in the phrase-concept pair can be gaps of length zero in the sentence).

Consider all phrase-concept pairs that match the input sentence. Let D be a graph of all the graph fragments in these pairs along with all labeled, directed edges between them. Let G be the predicted graph, which will be a subset of D.

Let $\mathbf{z}$ be a binary vector with n components, where each component of $\mathbf{z}$ indicates the presence or absence of a particular graph fragment or edge in G. To show the output dependence of G on $\mathbf{z}$ I will sometimes write $G_{\mathbf{z}}$. The decoding problem is expressed as a binary program with linear constraints, a connected constraint, and a non-linear scoring function (MCG):

$$\arg\max_{\mathbf{z}\in\{0,1\}^n} \mathit{score}(\mathbf{z}) \text{ s.t. } \mathbf{Az} \leq \mathbf{b} \text{ and } G_{\mathbf{z}} \text{ is weakly connected} \tag{14}$$

The constraints $\mathbf{Az} \leq \mathbf{b}$ are used to ensure that each word is only translated once, edge labels ARG0-ARGN are deterministic, and "and" concepts have at least two children. To solve Eq. 14 approximately, I use randomized greedy inference which will be discussed in the next section.

4.5 Maximum connected subgraph with linear constraints and a global scoring function (MCG)

I will solve Eq. 14, MCG, approximately using an algorithm based on hill-climbing with random restarts. The pseudo-code for the algorithm is given in Algorithms 2-6. The input to the algorithm is the dense graph D, the constraints $\mathbf{Az} \leq \mathbf{b}$, the scoring function score, a maxdepth parameter, as well as a function G_0 to provide random initial graphs that are connected and satisfy the constraints. In general, with arbitrary constraints, even finding a single graph that satisfies the constraints is NP-hard.[12] The decoding algorithm relies on a domain specific algorithm to provide the initial graph. For cross-lingual parsing, this graph will be found by randomly adding fragments from D with edges between them satisfying the constraints until no more fragments can be added.

Here is an overview of the hill-climbing algorithm. Starting with $G = G_0$, graph fragments and edges are randomly added to G to see if they improve the score. Adding or deleting a graph fragment or edges is called an **action**. If an action would cause a violation in the constraints, a targeted depth first search (DFS) is performed, adding or removing other graph fragments and edges to satisfy the constraints[13]. If an action provides no improvement to the score after DFS, or if DFS cannot find a graph satisfying the constraints after searching to a certain depth, then the action is not performed. Otherwise the action is performed on G and the program repeats. If there is no action that improves the score, then hill-climbing returns G. The hill-climbing process is repeated many times with different initial graphs (random restarts), and the highest scoring graph is returned.

4.6 Evaluation and experiments

For the experiments I will use the Chinese-English FBIS corpus as training data. To evaluate, I will compare two different ways: 1) using automatic parses on the standard MT test sets, using the four references to

[12] This is because any binary integer linear program can be written as Eq. 14 by having a node in D for each variable, and adding an extra node that is connected to every other node with a single edge.

[13] During DFS, the algorithm keeps track of the initial action and actions that are performed during DFS so it doesn't undo or repeat actions.

produce four separate test examples each input source sentence and 2) using gold standard AMRs on the MT09 test set (200 sentences).

4.7 Status

This section is proposed work. Some of the supporting code has been written.

5 Timeline

I give two timelines for the thesis. The first timeline (Fig. 5) is an aggressive timeline that includes more than I have outlined in this thesis proposal: applying the randomized greedy inference algorithm from §4 to mono-lingual parsing and to generation by rewriting the generator as a MCG problem and applying the graph algorithm from §4.5 for global decoding with arbitrary features. The timeline is:

- April 11 - June 3: global cross-lingual parsing (paper goal: EMNLP, June 3)
- June 3 - August 15: global mono-lingual parsing (paper goal: TACL Sept.)
- August 16 - November 15: global generation (paper goal: ACL 2017)
- October 1 - January 1, 2017: thesis writing

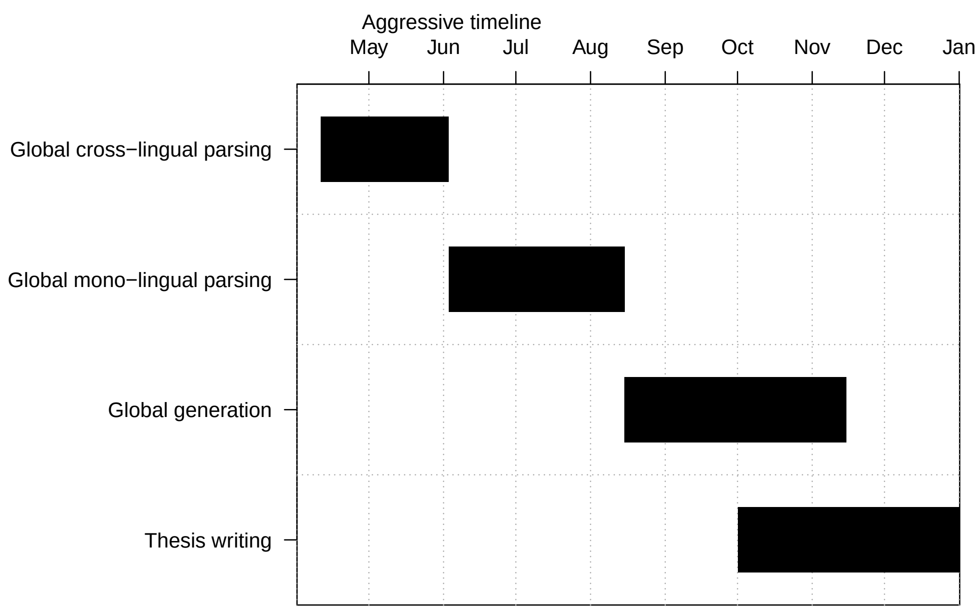


Figure 5: Aggressive timeline.

A more conservative timeline is (see Fig. 6):

- April 11 - June 3: global cross-lingual parsing (paper goal: TACL Aug.)
- June 3 - August 15: global mono-lingual parsing (paper goal: ACL 2017)
- October 1 - January 1, 2017: thesis writing

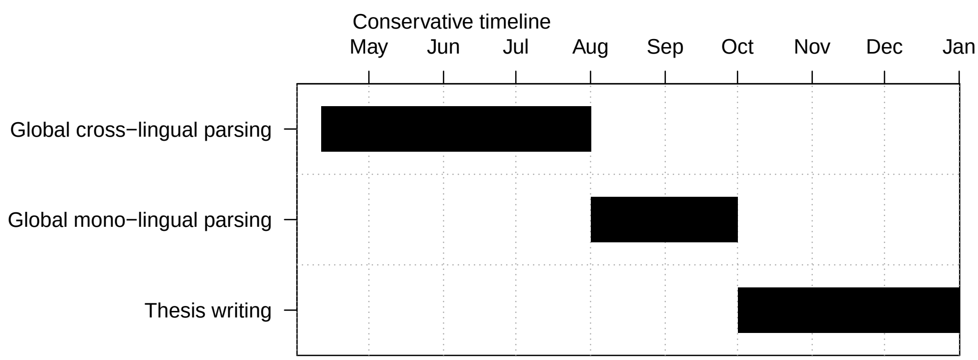


Figure 6: Conservative timeline.

6 Appendix: Algorithms for solving MCG

```
Function HillClimb(D, z, A, b, G0, score, maxdepth)
    input : directed graph D,
            nodes and edges of D labeled with binary vector z,
            constraint Az ≤ b,
            initial graph G0 (subgraph of D) which is weakly connected and satisfies the constraints,
            scoring function score,
            maximum depth of depth-first search maxdepth
            all input parameters are global variables (visible in called functions)
    output: maximum scoring weakly connected subgraph of D that satisfies Az ≤ b
    G = G0;
    best_score = score(G);
    updated = true;
    while updated do
        updated = false;
        for action in add nodes not in G, add edges between nodes in G, remove edges in G,
        remove nodes in G, sorted randomly do
            G = action applied to G;
            if G violates Az ≤ b or an edge or concept was deleted and G is not weakly connected
            then
                applied_actions = {action};
                G = DFS (G, applied_actions, best_score, maxdepth) ;
            end
            if G is weakly connected and does not violate Az ≤ b and score(G) > best_score
            then
                best_score = score(G);
                updated = true;
            else
                G = inverse of action applied to G;
            end
        end
    end
    return G;
end
```

Algorithm 2: Hill climb algorithm.

```
Function DFS (G, applied_actions, best_score, depth)
    input  : directed graph G (subgraph of D),
             applied_actions,
             best_score,
             depth,
             all parameters of HillClimb
    output: subgraph of D that may or may not be weakly connected or satisfy Az ≤ b

    new_best_score = best_score;
    G_best = G;
    if depth = 0 then
        return G;
    end
    if if there is a violated constraint which involves only one node or only one edge then /* this
    constraint can't be fixed                                                                  */
        return G;
    end
    for actions in violated_no_alt (applied_actions) do
        G = action applied to G;
    end
    if G is weakly connected and Az ≤ b and score(G) > new_best_score then
        new_best_score = score(G);
        G_best = G;
    end
    for action in violated_connected (applied_actions) ∪ violated_Az_b
    (applied_actions) do
        G_new = DFS (G with action applied, applied_actions∪{action}, best_score,
        depth −1);
        if if G_new is weakly connected and Az ≤ b and score(G_new) > new_best_score then
            new_best_score = score(G_new);
            G_best = G_new;
        end
    end
    return G_best;
end
```

Algorithm 3: DFS

Function `violated_Az_b(`G`, applied_actions)`

  input : directed graph G (subgraph of D), `applied_actions`, all parameters of `HillClimb`

  output: set of actions

```
actions= {};
for i in [0, length of z − 1] do
    if (Az)_i > b_i then /* violated constraint */
        for j in [0, length of z − 1] do
            if A_ij > 0 and z_j = 1 and j is not in applied_actions then
                add remove (j) to actions;
            end
            if A_ij < 0 and z_j = 0 and j is not in applied_actions then
                add add (j) to actions;
            end
        end
    end
end
return actions;
end
```

Algorithm 4: Returns the set of actions that may help resolve violated constraints.

Function `violated_no_alt(`G`, applied_actions)`

  input : directed graph G (subgraph of D), `applied_actions`, all parameters of `HillClimb`

  output: set of actions

```
actions= {};
for i in [0, length of z − 1] do
    if (Az)_i > b_i and row i of A has exactly one non-zero element j that is not in
    applied_actions then
        if A_ij > 0 and z_j = 1 and j is not in applied_actions then
            add remove (j) to actions;
        end
        if A_ij < 0 and z_j = 0 and j is not in applied_actions then
            add add (j) to actions;
        end
    end
end
return actions;
end
```

Algorithm 5: Returns the set of actions that must be applied to resolve violated constraints (i.e. there are no alternatives).

```
Function violated_connected(G, applied_actions)
    input  : directed graph G (subgraph of D),
             applied_actions,
             all parameters of HillClimb
    output: set of actions
    actions= {};
    added_nodes = nodes which have been added in applied_actions;
    active_nodes = nodes which are the endpoints of deleted edges in applied_actions;
    if Nodes(G)∩active_nodes= {}:
        for each connected component C of G:
            for added_node in Nodes(C)∩added_nodes:
                for node in Nodes(G) − Nodes(C):
                    for edge in D which connects added_node and node:
                        add add(edge) to actions
    elif Nodes(G)∩added_nodes≠ {}:
        for each connected component C of G:
            for active_node in Nodes(C)∩active_nodes:
                for added_node in (Nodes(G) − Nodes(C))∩added_nodes:
                    for edge in D which connects active_node and added_node and the label
                    of edge matches the label of a deleted edge to or from active_node:
                        add add(edge) to actions
    if actions = {}:
        for each connected component C of G:
            for active_node in Nodes(C)∩active_nodes:
                for node in Nodes(G) − Nodes(C):
                    added = false;
                    if node is not in active_nodes:
                        for edge in D which connects active_node and node and the label of
                        edge matches the label of a deleted edge to or from active_node:
                            add add(edge) to actions;
                            added = true;
                    if added = false:
                        for edge in D which connects active_node and node:
                            add add(edge) to actions;
    return actions;
```

Algorithm 6: Returns the set of actions that help resolve a violated connected constraint.

References

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Banarescu et al., 2013] Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186. Association for Computational Linguistics.

[Cai and Knight, 2013] Cai, S. and Knight, K. (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)*, page 748–752. Association for Computational Linguistics.

[Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8. Association for Computational Linguistics.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[de Marneffe et al., 2006] de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, pages 449–454.

[Dorr et al., 1998] Dorr, B., Habash, N., and Traum, D. (1998). A thematic hierarchy for efficient generation from lexical-conceptual structure. In Farwell, D., Gerber, L., and Hovy, E., editors, *Machine Translation and the Information Soup: Proceedings of the Third Conference of the Association for Machine Translation in the Americas*, number 1529 in Lecture Notes in Computer Science, pages 333–343. Springer.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[Dyer et al., 2010] Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Ture, F., Blunsom, P., Setiawan, H., Eidelman, V., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proc. of ACL*.

[Flanigan et al., 2014] Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of ACL*.

[Galley et al., 2004] Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule? In *Proc. of HLT-NAACL*.

[Goodman, 1999] Goodman, J. (1999). Semiring parsing. *CL*, 25(4).

[Graehl and Knight, 2004] Graehl, J. and Knight, K. (2004). Training tree transducers. In Susan Dumais, D. M. and Roukos, S., editors, *HLT-NAACL 2004: Main Proceedings*, pages 105–112, Boston, Massachusetts, USA. Association for Computational Linguistics.

[Heafield et al., 2013] Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proc. of ACL*.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *JAIR*, 14(1):253–302.

[Huang et al., 2006] Huang, L., Knight, K., and Joshi, A. (2006). Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*.

[Jones et al., 2012] Jones, B., Andreas, J., Bauer, D., Hermann, K. M., and Knight, K. (2012). Semantics-based machine translation with hyperedge replacement grammars. In *COLING*, pages 1359–1376.

[Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430. Association for Computational Linguistics.

[Knight et al., 2016] Knight, K., Hermjakob, U., Griffitt, K., Palmer, M., Bonial, C., O'Gorman, T., Schneider, N., Badarau, B., and Bardocz, M. (2016). Deft phase 2 amr annotation r2 ldc2016e25. Linguistic Data Consortium.

[Kruskal, 1956] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48.

[Liu et al., 2015] Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1077–1086, Denver, Colorado. Association for Computational Linguistics.

[Melcuk, 1988] Melcuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, NY, USA.

[Mitra and Baral, 2016] Mitra, A. and Baral, C. (2016). Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. *AAAI*.

[Och, 2003] Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proc. of ACL*.

[Palmer et al., 2005] Palmer, M., Gildea, D., and Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

[Ratinov and Roth, 2009] Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155. Association for Computational Linguistics.

[Rosenblatt, 1957] Rosenblatt, F. (1957). The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory.

[Taskar et al., 2003] Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. MIT Press.

[Tsochantaridis et al., 2004] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 104–, New York, NY, USA. ACM.

[Yu and Joachims, 2009] Yu, C.-N. J. and Joachims, T. (2009). Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1169–1176, New York, NY, USA. ACM.

[Zhang et al., 2014] Zhang, Y., Lei, T., Barzilay, R., and Jaakkola, T. (2014). Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1013–1024, Doha, Qatar. Association for Computational Linguistics.

[Zhang et al., 2015] Zhang, Y., Li, C., Barzilay, R., and Darwish, K. (2015). Randomized greedy inference for joint segmentation, pos tagging and dependency parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 42–52, Denver, Colorado. Association for Computational Linguistics.