# CLIP@UMD at SemEval-2016 Task 8: Parser for Abstract Meaning Representation using Learning to Search

**Sudha Rao**[1,3]*, **Yogarshi Vyas**[1,3]*, **Hal Daumé III**[1,3], **Philip Resnik**[2,3]
[1]Computer Science, [2]Linguistics, [3]UMIACS
University of Maryland
{raosudha,yogarshi,hal}@cs.umd.edu, resnik@umd.edu

## Abstract

In this paper we describe our approach to the Abstract Meaning Representation (AMR) parsing shared task as part of SemEval 2016. We develop a novel technique to parse English sentences into AMR using Learning to Search. We decompose the AMR parsing task into three subtasks - that of predicting the concepts, the relations, and the root. Each of these subtasks are treated as a sequence of predictions. Using Learning to Search, we add past predictions as features for future predictions, and define a combined loss over the entire AMR structure.

## 1 Introduction

This paper describes our submission to the Abstract Meaning Representation (AMR) Parsing Shared Task at SemEval 2016. The goal of the task is to generate AMRs automatically for English sentences. We develop a novel technique for AMR parsing that uses Learning to Search (L2S) (Ross et al., 2011; Daumé III et al., 2009; Collins and Roark, 2004).

L2S is a family of approaches that solves structured prediction problems. These algorithms have proven to be highly effective for problems in NLP like part-of-speech tagging, named entity recognition (Daumé III et al., 2014), coreference resolution (Ma et al., 2014), and dependency parsing (He et al., 2013). Briefly, L2S attempts to do structured prediction by (1) decomposing the production of the structured output in terms of an explicit search space (states, actions, etc.); and (2) learning hypotheses that control a policy that takes actions in this search space. AMR (Banarescu et al., 2013), in turn, is a structured semantic representation which is a rooted, directed, acyclic graph. The nodes of this graph represent concepts in the given sentence and the edges represent relations between these concepts. As such, the task of predicting AMRs can be naturally placed in the L2S framework. This allows us to model the learning of concepts and relations in a unified setting which aims to minimize the loss over the entire predicted structure.

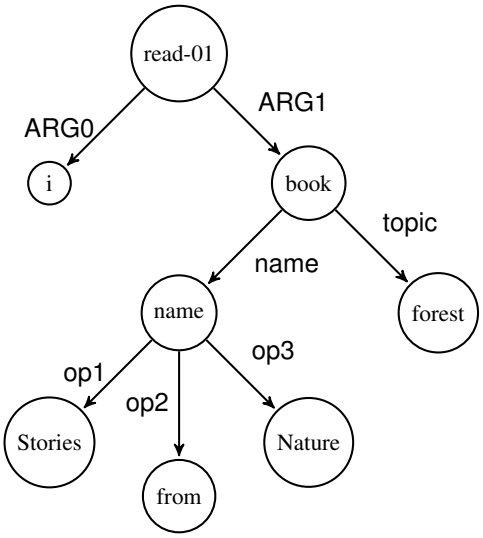


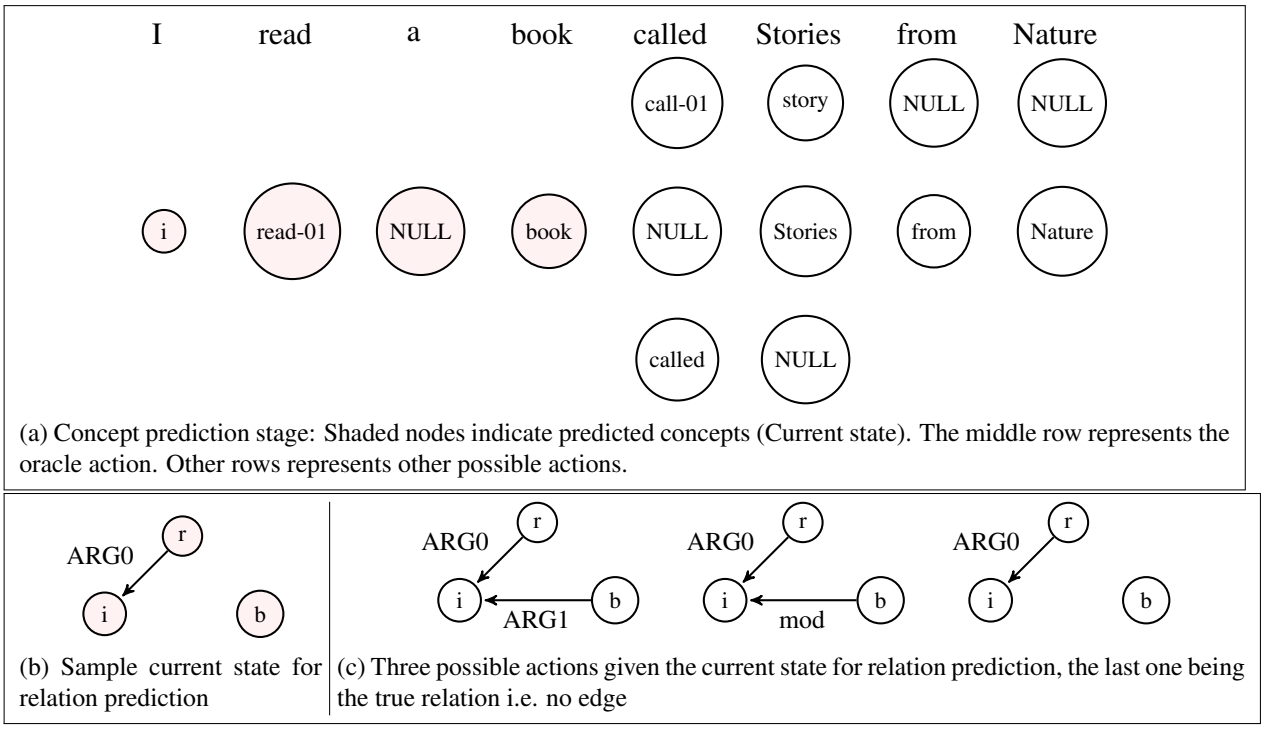

Figure 1: AMR graph for the sentence *"I read a book, called Stories from Nature, about the forest."*

In the next section, we briefly review DAGGER and explain its various components with respect to our AMR parsing task. Section 3 describes our main algorithm along with the strategies we use to deal with the large search space of the search problem.

*The first two authors contributed equally to this work.



*Proceedings of SemEval-2016*, pages 1081–1087,
San Diego, California, June 16-17, 2016. 




(a) Concept prediction stage: Shaded nodes indicate predicted concepts (Current state). The middle row represents the oracle action. Other rows represents other possible actions.

(b) Sample current state for relation prediction

(c) Three possible actions given the current state for relation prediction, the last one being the true relation i.e. no edge

Figure 2: Using DAGGER for AMR parsing

We then describe our experiments and results (Section 4).

## 2 Using L2S for AMR Parsing

L2S works on the notion of a policy which can be defined as "what is the best next action ($y_i$) to take" in a search space given the current state. It starts with an initial policy on a trajectory (called rollin policy), takes a one-step deviation and completes the trajectory with another policy (called the rollout policy). The different variations of L2S are defined based on what kind of policies it uses during rollin and rollout. For example DAGGER uses rollin=learned policy and rollout=reference policy, SEARN uses rollin=rollout=stochastic mixture of reference and learned policy, LOLS uses rollin=learned policy and rollout=stochastic mixture of reference and learned policy.

Next, we describe how we use L2S for AMR parsing. We decompose the full AMR parsing task into three subtasks - that of predicting the concepts, predicting the root, and predicting the relations between the predicted concepts (explained in more detail under section 3). The search space for concept and relation prediction consists of a state $s = (x_1, x_2, ..., x_m, y_1, y_2, .., y_{i-1})$, where the input $(x_1, x_2, ..., x_m)$ are the $m$ words of a sentence.

- During concept prediction, the labels $(y_1, y_2, .., y_{i-1})$ are the concepts predicted up to the index $(i-1)$ and the next action $y_i$ is the concept for word $x_i$ from a $k$-best list of concepts. In Figure 2a, the current state corresponds to the concepts {'i', 'read-01', 'book'} and the next action is assigning one of {'call-01', 'called', NULL} to the word 'called'.

- During relation prediction, the labels are the relations predicted for pairs of concepts obtained during the concept prediction stage. In Figure 2c, the current state corresponds to the relation 'ARG0' predicted between 'r' and 'i' and the next action is assigning one of {'ARG1', 'mod', NO-EDGE} to the pair of concept 'b' and 'i'.

For the root prediction subtask we train a multi-class classifier which makes a single prediction - that

of choosing the root concept from all the predicted concepts.

Next, we need to define how we learn a policy for AMR parsing. When the reference policy is optimal, it has been shown that it is effective to rollout using the reference policy (Ross and Bagnell, 2014; Chang et al., 2015). This is true for our task and hence we use DAGGER. Below, we explain how we use DAGGER, with a running example in Figure 2.

At training time, DAGGER operates in an iterative fashion. It starts with an initial policy and given an input $x$, makes a prediction $y = y_1, y_2, ..., y_m$ using the current policy. For each prediction $y_i$ it generates a set of multi-class classification examples each of which correspond to a possible action the algorithm can take given the current state. Each example can be defined using local features and features that depend on previous predictions. We use a one-against-all classifier to make the multi-class classification decisions.

During training, DAGGER has access to the reference policy. The reference policy during concept prediction is to predict the concept that was aligned to a given span using the JAMR aligner (explained in Section 3.4) For e.g. in Figure 2a, the reference policy is to predict NULL since the span "called" was aligned to NULL by the aligner. The reference policy during relation prediction is to predict the gold edge between two concepts. For e.g. in Figure 2c, the reference policy is to predict NO-EDGE.

DAGGER then calculates the loss between the predicted action and the best action using a pre-specified loss function. It then computes a new policy based on this loss and interpolates it with the current policy to get an updated policy, before moving on to the next iteration. At test time, predictions are made greedily using the policy learned during training.

## 3 Methodology

### 3.1 Learning technique

We use DAGGER as described in Section 2 to learn a model that can successfully predict the AMR $y$ for a sentence $x$. The sentence $x$ is composed of a sequence of spans $(s_1, s_2, ..., s_n)$ each of which can be a single word or a span of words (We describe how we go from a raw sentence to a sequence of spans in Section 3.4). Given that our input has $n$ spans, we first decompose the structure into a sequence of $n^2 + 1$ predictions $\mathbf{D} = (\mathbf{C}, \mathbf{ROOT}, \mathbf{R})$, where

**Algorithm 1**

```
for each span s_i do
    c_i = predict_concept(s_i)
end for
c_root = predict_root([c_1, ..., c_n])
for each concept c_i do
    for each j < i do
        r_(i,j) = predict_relation(c_i, c_j)
        r_(j,i) = predict_relation(c_j, c_i)
    end for
end for
```

$\mathbf{C} = c_1, c_2, ..., c_n$ - where $c_i$ is the concept predicted for span $s_i$

**ROOT** is the decision of choosing one of the predicted concepts as the root ($c_{root}$) of the AMR

$\mathbf{R} = r_{2,*}, r_{*,2}, r_{3,*}, r_{*,3}, ..., r_{n,*}, r_{*,n}$ - where $r_{i,*}$ are the predictions for the directed relations from $c_i$ to $c_j$ $\forall j < i$, and $r_{*,i}$ are the predictions for the directed relations from $c_j$ to $c_i$ $\forall j < i$. We constrain our algorithm to not predict any incoming relations to $c_{root}$.

During training time, the possible set of actions for each prediction is given by the $k$-best list (Section 3.2). We use Hamming Loss as our loss function. Under Hamming Loss, the reference policy is simply choosing the right action for each prediction i.e. choosing the correct concept (relation) during the concept (relation) prediction phase. This loss is defined on the entire predicted output, and hence the model learns to minimize the loss for concepts and relations jointly.

Algorithm 1 describes the sequence of predictions to be made in our problem. We learn three different policies corresponding to each of the functions $predict\_concept$, $predict\_root$ and $predict\_relation$. The learner in each stage uses features that depend on predictions made in the previous stages. Tables 1, 2 and 3 describe the set of features we use for the concept prediction, relation prediction and root prediction stages respectively.

| Feature label | Description |
|---|---|
| $w_{i-2}, w_{i-i}, w_i, w_{i+1}, w_{i+2}$ | Words in $s_i$ and context |
| $p_{i-2}, p_{i-i}, p_i, p_{i+1}, p_{i+2}$ | POS tags of words in $s_i$ and context |
| $NE_i$ | Named entity tags for words in $s_i$ |
| $s_i$ | Binary feature indicating whether $w_i$ is(are) stopword(s) |
| $dep_i$ | All dependency edges originating from words in $w_i$ |
| $b_c$ | Binary feature indicating whether $c$ is the most frequently aligned concept with $s_i$ or not |
| $c_{i-2}, c_{i-1}$ | Predicted concepts for two previous spans |
| $c$ | Concept label and its conjunction with all previous features |
| $frame_i$ and $sense_i$ | If the label is a PropBank frame (e.g. 'see-01', use the frame ('see') and the sense('01') as additional features. |

Table 1: Concept prediction features for span $s_i$ and concept label $c_i$

| Feature label | Description |
|---|---|
| $c_i, c_j, c_i \wedge c_j$ | The two concepts and their conjunction |
| $w_i, w_j, w_i \wedge w_j$ | Words in the corresponding spans and their conjunction |
| $p_i, p_j, p_i \wedge p_j$ | POS tags of words in spans and their conjunction |
| $dep_{ij}$ | All dependency edges with tail in $w_i$ and head in $w_j$ |
| $dir$ | Binary feature which is true iff $i < j$ |
| $r$ | Relation label and its conjunction with all other features |

Table 2: Relation prediction features for concepts $c_i$ and $c_j$ and relation label $r$

| Feature label | Description |
|---|---|
| $c_i$ | Concept label. If the label is a PropBank frame (e.g. 'see-01', use the frame ('see') and the sense('01') as additional features. |
| $w_i$ | Words in $s_i$, i.e. the span corresponding to $c_i$ |
| $p_i$ | POS tags of words in $s_i$ |
| $is\_dep\_root_i$ | Binary feature indicating whether one of the words in $s_i$ is the root in the dependency tree of the sentence |

Table 3: Root prediction features for concept $c_i$

### 3.2 Selecting $k$-best lists

For predicting the concepts and relations using DAGGER, we need a candidate-list (possible set of actions) to make predictions from.

**Concept candidates:** For a span $s_i$, the candidate-list of concepts, CL-CON$_{s_i}$ is the set of all concepts that were aligned to $s_i$ in the entire training data. If $s_i$ has not been seen in the training data, CL-CON$_{s_i}$ consists of the lemmatized span, PropBank frames (for verbs) obtained using the Unified Verb Index (Schuler, 2005) and the NULL concept.

**Relation candidates:** The candidate list of relations for a relation from concept $c_i$ to concept $c_j$, CL-REL$_{ij}$, is the union of the following three sets:

- $pairwise_{i,j}$ - All directed relations from $c_i$ to $c_j$ when $c_i$ and $c_j$ occurred in the same AMR,
- $outgoing_i$ - All outgoing relations from $c_i$, and
- $incoming_j$ - All incoming relations into $c_j$.

In the case when both $c_i$ and $c_j$ have not been seen in the training data, CL-REL$_{ij}$ consists of all relations seen in the training data. In both cases, we also provide an option NO-EDGE which indicates that there is no relation between $c_i$ and $c_j$.

### 3.3 Pruning the search space

To prune the search space of our learning task, and to improve the quality of predictions, we use two observations about the nature of the edges of the AMR

of a sentence, and its dependency tree (obtained using the Stanford dependency parser (De Marneffe et al., 2006)), within our algorithm.

First, we observe that a large fraction of the edges in the AMR for a sentence are between concepts whose underlying spans (more specifically, the words in these underlying spans) are within two edges of each other in the dependency tree of the sentence. Thus, we refrain from calling the $predict\_relation$ function in Algorithm 1 between concepts $c_i$ and $c_j$ if each word in $w_i$ is three or more edges away from all words in $w_j$ in the dependency tree of the sentence under consideration, and vice versa. This implies that there will be no relation $r_{ij}$ in the predicted AMR of that sentence. This doesn't affect the number of calls to $predict\_relation$ in the worst case ($n^2 - n$, for a sentence with $n$ spans), but practically, the number of calls are far fewer. Also, to make sure that this method does not filter out too many AMR edges, we calculated the percentage of AMR edges that are more than two edges away in dependency tree. We found this number to be only about 5% across all our datasets.

Secondly, and conversely, we observe that for a large fraction of words which have a dependency edge between them, there is an edge in the AMR between the concepts corresponding to those two words. Thus, when we observe two concepts $c_i$ and $c_j$ which satisfy this property, we force our $predict\_relation$ function to assign a relation $r_{ij}$ that is not NULL.

### 3.4 Preprocessing

**JAMR Aligner**: The training data for AMR parsing consists of sentences paired with corresponding AMRs. To convert a raw sentence into a sequence of spans (as required by our algorithm), we obtain alignments between words in the sentence and concepts in the AMR using the automatic aligner of JAMR. The alignments obtained can be of three types (Examples refer to Figure 1):

- *A single word aligned to a single concept*: E.g., word 'read' aligned to concept 'read-01'.
- *Span of words aligned to a graph fragment*: E.g., span 'Stories from Nature' aligned to the graph fragment rooted at 'name'. This usually happens for named entities and multiword expressions such as those related to date and time.
- *A word aligned to* NULL *concept*: Most function words like 'about', 'a', 'the', etc are not aligned to any particular concept. These are considered to be aligned to the NULL concept.

**Forced alignments**: The JAMR aligner does not align all concepts in a given AMR to a span in the sentence. We use a heuristic to forcibly align these leftover concepts and improve the quality of alignments. For every unaligned concept, we count the number of times an unaligned word occurs in the same sentence with the unaligned concept across all training examples. We then align every leftover concept in every sentence with the unaligned word in the sentence with which it has maximally coocurred.

**Span identification**: During training time, the aligner takes in a sentence and its AMR graph and splits each sentence into spans that can be aligned to the concepts in the AMR. However, during test time, we do not have access to the AMR graph. Hence, given a test sentence, we need to split the sentence into spans, on which we can predict concepts. We consider each word as a single span except for two cases. First, we detect possible multiword spans corresponding to named entities, using a named entity recognizer (Lafferty et al., 2001). Second, we use some basic regular expressions to identify time and date expressions in sentences.

### 3.5 Connectivity

Algorithm 1 does not place explicit constraints on the structure of the AMR. Hence, the predicted output can have disconnected components. Since we want the predicted AMR to be connected, we connect the disconnected components (if any) using the following heuristic. For each component, we find its roots (i.e. concepts with no incoming relations). We then connect the components together by simply adding an edge from our predicted root $c_{root}$ to each of the component roots. To decide what edge to use between our predicted root $c_{root}$ and the root of a component, we get the $k$-best list (as described in section 3.2) between them and choose the most frequent edge from it.

| Dataset | Training | Dev | Test |
|---|---|---|---|
| BOLT DF MT | 1061 | 133 | 133 |
| Broadcast conversation | 214 | 0 | 0 |
| Weblog and WSJ | 0 | 100 | 100 |
| BOLT DF English | 6455 | 210 | 229 |
| Guidelines AMRs | 689 | 0 | 0 |
| 2009 Open MT | 204 | 0 | 0 |
| Proxy reports | 6603 | 826 | 823 |
| Weblog | 866 | 0 | 0 |
| Xinhua MT | 741 | 99 | 86 |

Table 4: Dataset statistics. All figures represent number of sentences.

### 3.6 Acyclicity

The post-processing step described in the previous section ensures that the predicted AMRs are rooted, connected, graphs. However, an AMR, by definition, is also acyclic. We do not model this constraint explicitly within our learning framework. Despite this, we observe that only a very small number of AMRs predicted using our fully automatic approach have cycles in them. Out of the total AMRs predicted in all test sets, less than 5% have cycles in them. Besides, almost all cycles that are predicted consist of only two nodes, i.e. both $r_{ij}$ and $r_{ji}$ have non-NO-EDGE values for concepts $c_i$ and $c_j$. To get an acyclic graph, we can greedily select one of $r_{ij}$ or $r_{ji}$, without any loss in parser performance.

## 4 Experiments and Results

The primary corpus for this shared task is the AMR Annotation Release 1.0 (LDC2015E86). This corpus consists of datasets from varied domains such as online discussion forums, blogs, and newswire, with about 19,000 sentence-AMR pairs. All datasets have a pre-specified training, dev and test split (Table 4).

We trained three systems. The first was trained on all available training data. The two other systems were trained using data from a single domain. Specifically, we chose *BOLT DF English* and *Proxy reports* since these are the two largest training datasets individually. The system trained on the *Proxy reports* dataset performed the best when evaluated on the dev set. Hence we used this system as our primary system for the task. Additionally, we use DAGGER as implemented in the Vowpal Wabbit machine learning library (Langford et al., 2007; Daumé III et al., 2014).

The evaluation of predicted AMRs is done using Smatch (Cai and Knight, 2013) [1], which compares two AMRs using precision, recall and $F_1$. Our system obtained a Smatch $F_1$ score of 0.46 with a $Precision$ of 0.51 and a $Recall$ of 0.43 on the test set in the Shared Task (We made a tokenization error during the actual semeval submission and so reported an $F_1$ score of 0.44 instead). The mean $F_1$ score of all systems submitted to the shared task was 0.55 and the standard deviation was 0.06.

## 5 Conclusion and Future work

We have presented a novel technique for parsing English sentences into AMR using DAGGER, a Learning to Search algorithm. We decompose the AMR parsing task into subtasks of concept prediction, root prediction and relation prediction. Using Learning to Search allows us to use past predictions as features for future predictions and also define a combined loss over the entire AMR structure. Additionally, we use a $k$-best candidate list constructed from the training data to make predictions from. To prune the large search space, we incorporate useful heuristics based on the dependency parse of the sentence. Our system is available for download [2].

Currently we ensure various properties of AMR, such as connectedness and acyclicity using heuristics. In the future, we plan to incorporate these as constraints in our learning technique.

## References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 748–752.

[1] http://amr.isi.edu/download/smatch-v2.0.tar.gz

[2] http://cs.umd.edu/~yogarshi/amr_parser.zip

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2058–2066.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Hal Daumé III, John Langford, and Stephane Ross. 2014. Efficient programmable learning to search. *arXiv preprint arXiv:1406.1837*.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1455–1464.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

John Langford, Lihong Li, and Alexander Strehl. 2007. Vowpal wabbit online learning project.

Chao Ma, Janardhan Rao Doppa, J Walker Orr, Prashanth Mannem, Xiaoli Fern, Tom Dietterich, and Prasad Tadepalli. 2014. Prune-and-score: Learning for greedy coreference resolution. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

Stéphane Ross, Geoff J. Gordon, and J. Andrew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Workshop on Artificial Intelligence and Statistics (AIStats)*.

Karin Kipper Schuler. 2005. Verbnet: A broad-coverage, comprehensive verb lexicon.