Imitation Learning of Agenda-based Semantic Parsers

Jonathan Berant
Stanford University
yonatan@cs.stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Abstract

Semantic parsers conventionally construct logical forms bottom-up in a fixed order, resulting in the generation of many extraneous partial logical forms. In this paper, we combine ideas from imitation learning and agenda-based parsing to train a semantic parser that searches partial logical forms in a more strategic order. Empirically, our parser reduces the number of constructed partial logical forms by an order of magnitude, and obtains a 6x-9x speedup over fixed-order parsing, while maintaining comparable accuracy.

1 Introduction

Semantic parsing, the task of mapping natural language to semantic representations (e.g., logical forms), has emerged in recent years as a promising paradigm for developing question answering systems (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Kwiatkowski et al., 2010; Liang et al., 2011) and other natural language interfaces (Zettlemoyer and Collins, 2007; Tellex et al., 2011; Matuszek et al., 2012). Recently, there have been two major trends: The first is to scale semantic parsing to large knowledge bases (KB) such as Freebase (Cai and Yates, 2013; Kwiatkowski et al., 2013; Berant and Liang, 2014). The second is to learn semantic parsers without relying on annotated logical forms, but instead on their denotations (answers) (Clarke et al., 2010; Liang et al., 2011); this lessens the annotation burden and has been instrumental in fueling the first trend (Berant et al., 2013).

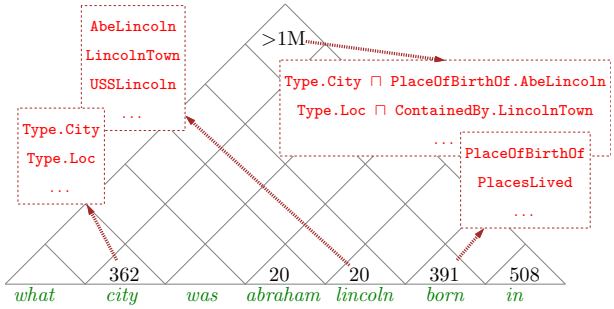

Figure 1: A parsing chart for the utterance *"what city was abraham lincoln born in"*. Numbers in chart cells indicate the number of possible semantic parses constructed over that span, and arrows point to some of the logical forms that were constructed. There are more than one million possible semantic parses for this utterance.

In this paper, we are interested in training semantic parsers from denotations on large KBs. The challenge in this setting is that the vocabulary of the target logical language often contains thousands of logical predicates, and there is a mismatch between the structure of the natural language and the logical language. As a result, the space of possible semantic parses for even a short utterance grows quickly. For example, consider the utterance *"what city was abraham lincoln born in"*. Figure 1 illustrates the number of possible semantic parses that can be constructed over some of the utterance spans. Just by combining semantic parses over the spans *"city"*, *"lincoln"* and *"born"* we already obtain $362 \cdot 391 \cdot 20$ possible parses; at the root, we get over a million parses.[1] The ambiguity of language thus results in a

[1] Even when type constraints are used to prune parses, we still produce more than a million possible parses at the root.

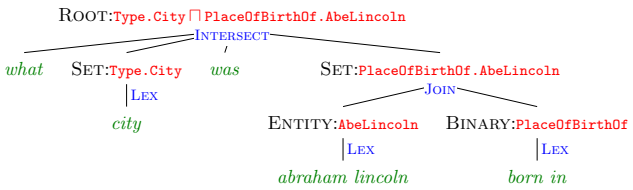

Figure 2: An example semantic parse, or *derivation*, for the utterance *"what city was abraham lincoln born in"*. Each node in the tree has a category (e.g., ENTITY) and a logical form (e.g., `AbeLincoln`).

hard search problem.

To manage this combinatorial explosion, past approaches (Krishnamurthy and Mitchell, 2012; Kwiatkowski et al., 2013; Berant et al., 2013) used beam search, where the number of parses (see Figure 2) for each chart cell (e.g., (SET, 3:5)) is capped at K. Typical bottom-up parsing is employed, where we build all parses for spans of length n before $n+1$, etc. This fixed-order parsing strategy constructs many unnecessary parses though. For example, it would create K parses for the category ENTITY and the span over *"lincoln"*, generating the logical form `USSLincoln`, although it is unlikely that this entity would be in the final logical form.

To overcome the problems with fixed-order parsing, we turn to *agenda-based parsing* (Kay, 1986; Caraballo and Charniak, 1998; Klein and Manning, 2003; Pauls and Klein, 2009; Auli and Lopez, 2011). In agenda-based parsing, an agenda (priority queue) holds partial parses that can be constructed next. At each step, the parse with the highest priority is popped from the agenda and put into the chart. This gives the parser full control over the sequence of parses constructed. But importantly, agenda-based parsing requires a good scoring function that can rank not just full parses but also partial parses on the agenda. How do we obtain such a scoring function?

To this end, we borrow ideas from imitation learning for structured prediction (Daume et al., 2009; Ross et al., 2011; Goldberg and Nivre, 2013; Chang et al., 2015). Specifically, we cast agenda-based semantic parsing as a Markov decision process, where the goal is to learn a *policy*, that given a *state* (i.e., the current chart and agenda), chooses the best next *action* (i.e., the parse to pop from the agenda). The supervision signal is used to generate a sequence of *oracle* actions, from which the model is trained.

Our work bears a strong resemblance to Jiang et al. (2012), who applied imitation learning to agenda-based parsing, but in the context of syntactic parsing. However, two new challenges arise in semantic parsing. First, syntactic parsing assumes gold parses, from which it is easy to derive an oracle action sequence. In contrast, we train from question-answer pairs only (rather than parse trees or even logical forms), so generating an oracle sequence is more challenging. Second, semantic parsers explore a much larger search space than syntactic parsers, due to the high level of uncertainty when translating to logical form. Thus, we hold a beam of parses for each chart cell, and modify learning for this setup.

We gain further efficiency by introducing a *lazy agenda*, which reduces the number of parses that need to be scored. For example, the single action of processing *"born"*, requires placing 391 logical forms on the agenda, although only few of them will be used. Our lazy agenda holds *derivation streams*, which implicitly represent a (possibly infinite!) group of related parses as a single agenda item, and lazily materialize parses as needed. Empirically, this reduces the number of parses that are scored at training time by 35%.

Last, we make modeling contributions by augmenting the feature set presented by Berant et al. (2013) with new features that improve the mapping of phrases to KB predicates.

We evaluate our agenda-based parser on the WEBQUESTIONS dataset (Berant et al., 2013) against a fixed-order parser, and observe that our parser reduces the number of parsing actions by an order of magnitude, achieves a 6x-9x speedup, and obtains a comparable accuracy of 49.7%.

To conclude, this paper describes three contributions: First, a novel agenda-based semantic parser that learns to choose good parsing actions, training from question-answer pairs only; Second, a lazy agenda that packs parses in streams and reduces the number of generated parses; Last, modeling changes that substantially improve accuracy.

2 Semantic Parsing Task

While our agenda-based semantic parser applies more broadly, our exposition will be based on our

primary motivation, question answering on a knowledge base. The semantic parsing task is defined as follows: Given (i) a knowledge base (KB) $\mathcal{K}$, (ii) a grammar G (defined shortly), and (iii) a training set of question-answer pairs $\{(x_i, y_i)\}_{i=1}^n$, output a semantic parser that maps new questions x to answers y via latent logical forms z.

We now briefly describe the KB and logical forms used in this paper. Let $\mathcal{E}$ denote a set of *entities* (e.g., `AbeLincoln`), and let $\mathcal{P}$ denote a set of *properties* (e.g., `PlaceOfBirthOf`). A *knowledge base* $\mathcal{K}$ is a set of *assertions* $(e_1, p, e_2) \in \mathcal{E} \times \mathcal{P} \times \mathcal{E}$ (e.g., (`Hodgenville`, `PlaceOfBirthOf`, `AbeLincoln`)). We use the Freebase KB (Google, 2013), which has 41M entities, 19K properties, and 596M assertions.

To query the KB, we use the logical language *simple λ-DCS*. In simple λ-DCS, an entity (e.g., `AbeLincoln`) denotes the singleton set containing that entity; this is a special case of a unary predicate. A property (a special case of a binary predicate) can be joined with a unary predicate; e.g., `PlaceOfBirthOf.AbeLincoln` denotes all entities that are the place of birth of Abraham Lincoln. We also have intersection: `Type.City` $\sqcap$ `PlaceOfBirthOf.AbeLincoln` denotes the set of entities that are both cities and the place of birth of Abraham Lincoln. We write $[\![z]\!]_{\mathcal{K}}$ for the denotation of a logical form z with respect to a KB $\mathcal{K}$. For a formal description of λ-DCS, see Liang (2013).

3 Grammars and Semantic Functions

Since we are learning semantic parsers from denotations, we cannot induce a grammar from provided logical forms (Kwiatkowski et al., 2010). Instead, we assume a small and flexible grammar that specifies the space of logical forms. The grammar consists of a backbone CFG, but is atypical in that each rule is augmented with a *semantic (composition) function* that produces a varying number of derivations using arbitrary context. This flexibility provides procedural control over the generation of logical forms.

Formally, a grammar is a tuple $(\mathcal{V}, \mathcal{N}, \mathcal{R})$, where $\mathcal{V}$ is a set of terminals (words), $\mathcal{N}$ is a set of categories (such as BINARY, ENTITY, SET and ROOT in Figure 2, where ROOT is the root category), and $\mathcal{R}$ is a rule set of binary and unary rules, explained below. A binary rule $r \in \mathcal{R}$ has the form $A \to B\,C\,[f]$, where $A \in \mathcal{N}$ is the left-hand-side, $B\,C \in \mathcal{N}^2$ is the right-hand-side (RHS), and f is a semantic function, explained below.

Given an utterance x, the grammar defines a set of *derivations* (semantic parse trees) over every span $x_{i:j} = (w_i, w_{i+1}, \ldots, w_{j-1})$. Define $\mathcal{D}$ to be the set of all derivations, and let $d^A_{i:j}$ be a derivation over the span $x_{i:j}$ of category A. Given the derivations $d^B_{i:k}$ and $d^C_{k:j}$ and the rule $r = A \to B\,C\,[f]$, the semantic function $f : \mathcal{D} \times \mathcal{D} \to 2^{\mathcal{D}}$ produces a set of derivations $f(d^B_{i:k}, d^C_{k:j})$ over $x_{i:j}$ with category A. In words, the semantic function takes two child derivations as input and produces a set of candidate output derivations. For each output derivation d, let $d.r$ be the rule used (SET $\to$ ENTITY BINARY[JOIN]) and $d.z$ be the logical form constructed by f, usually created by combining the logical forms of the child derivations (`PlaceOfBirthOf.AbeLincoln`). This completes our description of binary rules; unary rules $A \to B\,[f]$ and lexical rules $A \to w\,[f]$ are handled similarly, where $w \in \mathcal{V}^+$ is a sequence of terminals.

Figure 3 demonstrates the flexibility of semantic functions. The JOIN semantic function takes a derivation whose logical form is a binary predicate, and a derivation whose logical form is a unary predicate, and performs a *join* operation. LEX takes a derivation representing a phrase and outputs many candidate derivations. INTERSECT takes two derivations and attempts to intersect their logical forms (as defined in Section 2). In this specific case, no output derivations are produced because the KB types for `Type.City` and `ReleaseDateOf.LincolnFilm` do not match.

In contrast with CFG rules for syntactic parsing, rules with semantic functions generate sets of derivations rather than a single derivation. We allow semantic functions to perform arbitrary operations on the child derivations, access external resources such as Freebase search API and the KB. In practice, our grammar employs 11 semantic functions; in addition to JOIN, LEX, and INTERSECT, we use BRIDGE, which implements the bridging operation (see Section 8) from Berant et al. (2013), as well as ones that recognize dates and filter derivations based on part-of-speech tags, named entity tags, etc.

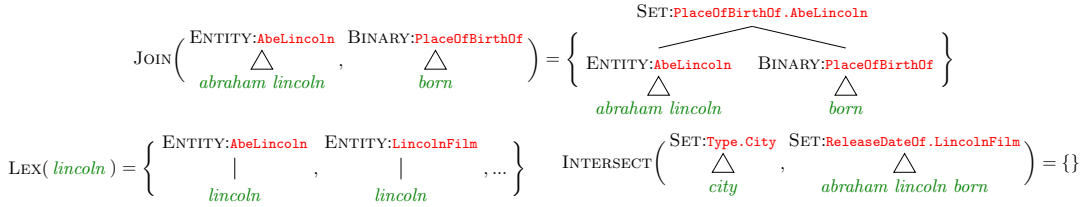

Figure 3: A semantic function (we show JOIN, LEX and INTERSECT) takes one or two child derivations and returns a set of possible derivations.

4 Fixed-order Parsing

We now describe fixed-order parsing with beam search, which has been the common practice in past work (Krishnamurthy and Mitchell, 2012; Kwiatkowski et al., 2013; Berant et al., 2013).

Let x be the input utterance. We call derivations $d^{\text{ROOT}}_{0:|x|}$, spanning the utterance x and with root category, *root derivations*, and all other derivations *partial derivations*. Given a scoring function $s : \mathcal{D} \to \mathbb{R}$, a bottom-up fixed-order parser iterates over spans $x_{i:j}$ of increasing length n and categories $A \in \mathcal{N}$, and uses the grammar to generate derivations based on derivations of subspans. We use beam search, in which for every span $x_{i:j}$ and every category A we keep a beam that stores up to K derivations in a *chart cell* $H^A_{i:j}$ (where different derivations usually correspond to different logical forms). We denote by H the set of derivations in any chart cell.

A fixed-order parser is guaranteed to compute the K highest-scoring derivations if the following two conditions hold: (i) all semantic functions return exactly one derivation, and (ii) the scoring function decomposes—that is, there is a function $s_{\text{rule}} : \mathcal{R} \to \mathbb{R}$ such that for every rule $r = A \to B\,C\,[f]$, the score of a derivation produced by the rule is $s(d^A_{i:j}) = s(d^B_{i:k}) + s(d^C_{k:j}) + s_{\text{rule}}(r)$. Unfortunately, the two conditions generally do not hold in semantic parsing. For example, the INTERSECT function returns an empty set when type-checking fails, violating condition (i), and the scoring function s often depends on the denotation size of the constructed logical form, violating condition (ii). In general, we want the flexibility of having the scoring function depend on the logical forms and sub-derivations, and therefore we will not be concerned with exactness in this paper. Note that we could augment the categories $\mathcal{N}$ with the logical form, but this would increase the number of categories exponentially.

Model. We focus on linear scoring functions: $s(d) = \phi(d)^\top \theta$, where $\phi(d) \in \mathbb{R}^F$ is the feature vector and $\theta \in \mathbb{R}^F$ is the parameter vector to be learned. Given any set of derivations $D \subseteq \mathcal{D}$, we can define the corresponding log-linear distribution:

$$p_\theta(d \mid D) = \frac{\exp\{\phi(d)^\top \theta\}}{\sum_{d' \in D} \exp\{\phi(d')^\top \theta\}}. \quad (1)$$

Learning. The training data consists of a set of utterance-denotation (question-answer) pairs $\{(x_i, y_i)\}_{i=1}^n$. To learn θ, we use an online learning algorithm, where on each (x_i, y_i), we use beam search based on the current parameters to construct a set of root derivations $D_i = H^{\text{ROOT}}_{0:|x|}$, and then take a gradient step on the following objective:

$$\mathcal{O}_i(\theta) = \log p(y_i \mid x_i) \quad (2)$$

$$= \log \sum_{d \in D_i} p_\theta(d \mid D_i) R(d) + \lambda \|\theta\|_1, \quad (3)$$

where $R(d) \in [0, 1]$ is a reward function that measures the compatibility of the predicted denotation $[\![d.z]\!]_{\mathcal{K}}$ and the true denotation y_i,[2]. We marginalize over latent derivations, which are weighted by their compatibility with the observed denotation y_i.

The main drawback of fixed-order parsing is that to obtain the K root derivations D_i, the parser must first construct K derivations for all spans and all categories, many of which will not make it into any root derivation $d \in D_i$. Next, we describe agenda-based parsing, whose goal is to give the parser better control over the constructed derivations.

[2] $[\![d.z]\!]_{\mathcal{K}}$ and y_i are both sets of entities, so R is the F_1 score.

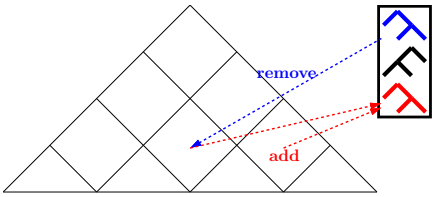

Figure 4: A schematic illustration of a executing a parsing action, specified by a derivation on the agenda. First, we *remove* it from the agenda and put it in the chart. Then, combine it with other chart derivations to produce new derivations, which are *added* back to the agenda.

Algorithm 1 Agenda-based parsing

procedure PARSE(x)
  INITAGENDA()
  while $|Q| > 0 \wedge |H^{\text{ROOT}}_{0:|x|}| < K$ **do**
    $d^A_{i:j} \leftarrow$ choose derivation from Q
    EXECUTEACTION($d^A_{i:j}$)
  choose and return derivation from $H^{\text{ROOT}}_{0:|x|}$
function EXECUTEACTION($d^A_{i:j}$)
  remove $d^A_{i:j}$ from Q
  if $|H^A_{i:j}| < K$ **then**
    $H^A_{i:j}$.add($d^A_{i:j}$)
    COMBINE($d^A_{i:j}$)
function COMBINE($d^A_{i:j}$)
  for $k > j$ and $r = B \to A\, C\, [f] \in \mathcal{R}$ **do**
    for $d^C_{j,k} \in H^C_{j:k}$ **do**
      Q.addAll($f(d^A_{i:j}, d^C_{j:k})$)
  for $k < i$ and $r = B \to C\, A\, [f] \in \mathcal{R}$ **do**
    for $d^C_{k,i} \in H^C_{k:i}$ **do**
      Q.addAll($f(d^C_{k:i}, d^A_{i:j})$)
function INITAGENDA()
  for $A \to x_{i:j}\, [f] \in \mathcal{R}$ **do**
    Q.addAll($f(x_{i:j})$)

5 Agenda-based Parsing

The idea of using an agenda for parsing has a long history (Kay, 1986; Caraballo and Charniak, 1998; Pauls and Klein, 2009). An agenda-based parser controls the order in which derivations are constructed using an agenda Q, which contains a set of derivations to be processed. At each point in time the *state* of the parser consists of two sets of derivations, the chart H and the agenda Q. Each *parsing action* chooses a derivation from the agenda, moves it to the chart, combines it with other chart derivations, and adds new derivations to the agenda (Figure 4).

Algorithm 1 describes agenda-based parsing. The algorithm shows binary rules; unary rules are treated similarly. First, we initialize the agenda by applying all rules whose RHS has only terminals, adding the resulting derivations to the agenda. Then, we perform parsing actions until either the agenda is empty or we obtain K root derivations. On each action, we first choose a derivation $d^A_{i:j}$ to remove from Q and add it to $H^A_{i:j}$, unless $H^A_{i:j}$ already has K derivations. Then, we combine $d^A_{i:j}$ with all derivations $d_{j:k}$ to the right and $d_{k:i}$ to the left. Upon termination, we perform a final action, in which we return a single derivation from all constructed root derivations.

The most natural way to choose an agenda derivation (and the root derivation in the final action) is by taking the highest scoring derivation $d = \arg\max_{d \in Q} s(d)$. Most work on agenda-based parsing generally assumed that the scoring function s is learned separately (e.g., from maximum likelihood estimation of a generative PCFG). Furthermore, they assumed that s satisfies the decomposition property (Section 4), which guarantees obtaining the highest scoring root derivation in the end. We, on the other hand, make no assumptions on s, and following Jiang et al. (2012), we *learn* a scoring function that is tightly coupled with agenda-based parsing. This is the topic of the next section.

6 Learning a Scoring Function

The objective in (2) is based on only a distribution over root derivations. Thus, by optimizing it, we do not explicitly learn anything about partial derivations that never make it to the root. Consider the derivation in Figure 1 over the phrase *"lincoln"* with the logical form `USSLincoln`. If none of the K root derivations contains this partial derivation, (2) will not penalize it, and we might repeatedly construct it even though it is useless. To discourage this, we need to be sensitive to intermediate parsing stages.

6.1 Imitation learning

We adapt the approach of Jiang et al. (2012) for agenda-based syntactic parsing to semantic parsing. Recall that a *parsing state* is $s = (H, Q)$, where $H \subseteq \mathcal{D}$ is the chart and $Q \subseteq \mathcal{D}$ is the agenda.[3]

The available actions are exactly the derivations

[3]To keep the state space discrete, states do not include derivation scores. This is why in Algorithm 1 we keep a list of up to K derivations in every chart cell rather than a beam, which would require actions to depend on derivation scores.

on the agenda Q, and the successor state s' is computed via EXECUTEACTION() from Algorithm 1. We model the *policy* as a log-linear distribution over (partial) agenda derivations Q: $p_\theta(a \mid s) = p_\theta(d = a \mid Q)$, according to (1). Note that the state s only provides the support of the distribution; the shape depends on only features $\phi(a)$ of the chosen action a, not on other aspects of s. This simple parameterization allows us to follow a policy efficiently: when we add a derivation a to the agenda, we insert it with priority equal to its score $s(a) = \phi(a)^\top \theta$. Computing the best action $\arg\max_a p_\theta(a \mid s)$ simply involves popping from the priority queue.

A history $\mathbf{h} = (s_1, a_1, \ldots, a_T, s_{T+1})$ (see Figure 5) is a sequence of states and actions, such that s_1 has an empty chart and an initial agenda, and s_{T+1} is a terminal state reached after performing the chart action in which we choose a root derivation a_T from $H^{\text{ROOT}}_{0:|x|}$ (Algorithm 1). The policy for choosing parsing actions induces a distribution over histories $p_\theta(\mathbf{h}) = \prod_{t=1}^{T} p_\theta(a_t \mid s_t)$.

At a high level, our policy is trained using imitation learning to mimic an oracle that takes a optimal action at every step (Daume et al., 2009; Ross et al., 2011). Because in semantic parsing we train from questions and answers, we do not have access to an oracle. Instead, we first parse x by sampling a history from the current policy p_θ; let d^* be the root derivation with highest reward out of the K root derivations constructed (see (2)). We then generate a target history $\mathbf{h}_{\text{target}}$ from d^* using two ideas—local reweighting and history compression, which we explain shortly. The policy parameters θ are then updated as follows:

$$\theta \leftarrow \theta + \eta\, R(\mathbf{h}_{\text{target}}) \sum_{t=1}^{T} \delta_t(\mathbf{h}_{\text{target}}), \quad (4)$$

$$\begin{aligned}\delta_t(\mathbf{h}) &= \nabla_\theta \log p_\theta(a_t \mid s_t) \quad (5)\\ &= \phi(a_t) - \mathbb{E}_{p_\theta(a'_t \mid s_t)}[\phi(a'_t)].\end{aligned}$$

The reward $R(\mathbf{h}) = R(a_T) \in [0, 1]$ measures the compatibility of the returned derivation (see (2)), and η is the learning rate.[4] Note that while our features $\phi(a)$ depend on the action only, the update rule takes into account all actions that are on the agenda.

[4] Note that unlike standard policy gradient, our updates are not invariant (even in expectation) to shifting the reward by a constant. Our updates do not maximize reward, but the reward merely provides a way to modulate the magnitude of the updates.

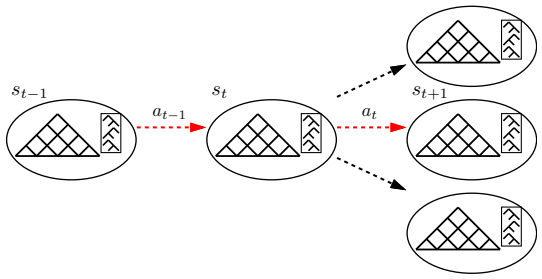

Figure 5: A schematic illustration of a (partial) history of states and actions. Each ellipse represents a state (chart and agenda), and the red path marks the actions chosen.

Local reweighting. Given the reference d^*, let $\mathbb{I}[a \text{ in } d^*]$ indicate whether an action a is a sub-derivation of d^*. We sample $\mathbf{h}_{\text{target}}$ from the locally reweighted distribution $p^{+\text{w}}_\theta(a \mid s) \propto p_\theta(a \mid s) \cdot \exp\{\beta\, \mathbb{I}[a \text{ in } d^*]\}$ for some $\beta > 0$. This is a multiplicative interpolation of the model distribution p_θ and the oracle. When β is high, this reduces to sampling from the available actions in d^*. When no oracle actions are available, this reduces to sampling from p_θ. The probability of a history is defined as $p^{+\text{w}}_\theta(\mathbf{h}) = \prod_{t=1}^{T} p^{+\text{w}}_\theta(a_t \mid s_t)$.

Recall we construct K root derivations. A problem with local reweighting is that after adding d^* to the chart, there are no more oracle actions on the agenda and all subsequent actions are simply sampled from the model. We found that updating towards these actions hurts accuracy. To avoid this problem, we propose performing *history compression*, described next.

History compression. Given d^*, we can define for every history $\mathbf{h}$ a sequence of indices $(t_1, t_2, \ldots)$ such that $\mathbb{I}[a_{t_i} \text{ in } d^*] = 1$ for every i. Then, the *compressed history* $c(\mathbf{h}) = (s_{t_1}, a_{t_1}, s_{t_2}, a_{t_2}, \ldots)$ is a sequence of states and actions such that all actions choose sub-derivations of d^*. Note that $c(\mathbf{h})$ is not a "real history" in the sense that taking action a_{t_i} does not necessarily result in state $s_{t_{i+1}}$. In Figure 6, the compressed history $c(\mathbf{h}) = (s_1, a_1, s_3, a_3, s_4, a_4, s_5)$.

We can now sample a target history $\mathbf{h}_{\text{target}}$ for (4) from a distribution over compressed histories,

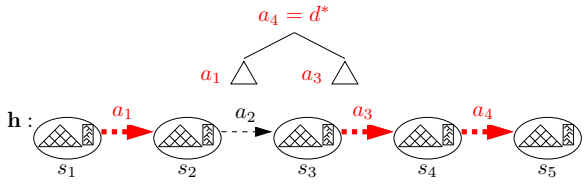

Figure 6: An example history of states and actions, where actions that are part of the reference derivation $d^* = a_4$ are in red. The compressed history is $c(\mathbf{h}) = (s_1, a_1, s_3, a_3, s_4, a_4, s_5)$.

Algorithm 2 Learning algorithm

procedure LEARN($\{x_i, y_i\}_{i=1}^n$)
  $\theta \leftarrow 0$
  for each iteration τ and example i **do**
    $\mathbf{h}_0 \leftarrow \text{PARSE}(p_\theta, x_i)$
    $d^* \leftarrow \text{CHOOSEORACLE}(\mathbf{h}_0)$
    $\mathbf{h}_{\text{target}} \leftarrow \text{PARSE}(p_\theta^{+\text{cw}}, x_i)$
    $\theta \leftarrow \theta + \eta_{\tau,i} \cdot R(\mathbf{h}_{\text{target}}) \sum_{t=1}^{T} \delta_t(\mathbf{h}_{\text{target}})$

$p_\theta^{+\text{c}}(\mathbf{h}) = \sum_{\mathbf{h}':c(\mathbf{h}')=\mathbf{h}} p_\theta(\mathbf{h})$, where we marginalize over all histories that have the same compressed history. To sample from $p_\theta^{+\text{c}}$, we sample $\mathbf{h}' \sim p_\theta$ and return $\mathbf{h}_{\text{target}} = c(\mathbf{h}')$. This will provide a history containing only actions leading to the oracle d^*. In our full model, we sample a history from $p_\theta^{+\text{cw}}$, which combines local reweighting and history compression: we sample $\mathbf{h}' \sim p_\theta^{+\text{w}}$ and return $\mathbf{h}_{\text{target}} = c(\mathbf{h}')$. We empirically analyze local reweighting and history compression in Section 9.

In practice, we set β large enough so that the behavior of $p_\theta^{+\text{cw}}$ is as follows: we first construct the reference d^* by sampling oracle actions. After constructing d^*, no oracle actions are on the agenda, so we construct $K-1$ more root derivations, sampling from p_θ (but note these actions are not part of the returned compressed history). Finally, the last action chooses d^* from the K derivations.

Algorithm 2 summarizes learning. We initialize our parameters to zero, and then parse each example by sampling a history from p_θ. We choose the derivation with highest reward in $H_{0:|x|}^{\text{ROOT}}$ as the reference derivation d^*. This defines $p_\theta^{+\text{cw}}$, which we sample from to update parameters. The learning rate $\eta_{\tau,i}$ is set using AdaGrad (Duchi et al., 2010).

6.2 Related approaches

Our method is related to policy gradient in reinforcement learning (Sutton et al., 1999): if in (4), we sample from the model distribution p_θ without an oracle, then our update is exactly the policy gradient update, which maximizes the expected reward $\mathbb{E}_{p_\theta(\mathbf{h})}[R(\mathbf{h})]$. We do not use policy gradient since the gradient is almost zero during the beginning of training, leading to slow convergence. This corroborates Jiang et al. (2012).

Our method extends Jiang et al. (2012) to semantic parsing, which poses the following challenges: (a) We train from denotations, and must obtain a reference to guide learning. (b) To combat lexical uncertainty we maintain a beam of size K in each parsing state (we show this is important in Section 9). (c) We introduce history compression, which focuses the learner on the actions that produce the correct derivation rather than incorrect ones on the beam. Interestingly, Jiang et al. (2012) found that imitation learning did not work well, and obtained improvements from interpolating with policy gradient. We found that imitation learning worked well, and interpolating with policy gradient did not offer further improvements. A possible explanation is that the uncertainty preserved in the K derivations in each chart cell allowed imitation learning to generalize properly, compared to Jiang et al. (2012), who had just a single item in each chart cell.

7 Lazy Agenda

As we saw in Section 3, a single semantic function (e.g., LEX, BRIDGE) can create hundreds of derivations. Scoring all these derivations when adding them to the agenda is wasteful, because most have low probability. In this section, we assume semantic functions return a *derivation stream*, i.e., an iterator that lazily computes derivations on demand. Our *lazy agenda* G will hold derivation streams rather than derivations, and the actual agenda Q will be defined only implicitly. The intuition is similar to lazy K-best parsing (Huang and Chiang, 2005), but is applied to agenda-based semantic parsing.

Our main assumption is that every derivation stream $g = [d_1, d_2, \ldots]$, is sorted by decreasing score: $s(d_1) \geq s(d_2) \geq \cdots$ (in practice, this is only approximated as we explain at the end of this section). We define the score of a derivation stream as $s(g) = s(d_1)$. At test time the only change to Algorithm 1 is in line 4, where instead of popping the

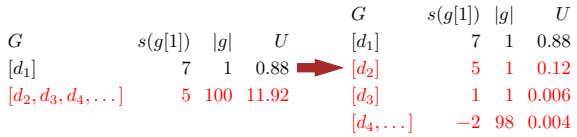

Figure 7: Unrolling a derivation where $\epsilon = 0.01$ and $|G^+| = 1$. The stream in red on the left violates the stopping condition, and so we unroll two derivations until all streams satisfy the condition.

highest scoring derivation, we pop the highest scoring derivation stream and process the first derivation on the stream. Then, we featurize and score the next derivation on the stream if the stream is not empty, and push the stream back to the agenda. This guarantees we will obtain the highest scoring derivation in every parsing action.

However, during training we *sample* from a distribution over derivations, not just return the argmax. Sampling from the distribution over streams can be quite inaccurate. Suppose the agenda contains two derivation streams: g_1 contains one derivation with score 1 and g_2 contains 50 derivations with score 0. Then we would assign g_1 probability $\frac{e^1}{e^1+e^0} = 0.73$ instead of the true model probability $\frac{e^1}{e^1+50e^0} = 0.05$. The issue is that the first derivation of g is not indicative of the actual probability mass in g.

Our solution is simple: before sampling (line 4 in Algorithm 1), we *process* the agenda to guarantee that the sum of probabilities of all unscored derivations is smaller than ϵ. Let G be the lazy agenda and $G^+ \subseteq G$ be the subset of derivation streams that contain more than one derivation (where unscored derivations exist). If for every $g \in G^+$, $p_\theta(g) = \sum_{d \in g} p_\theta(d) \leq \frac{\epsilon}{|G^+|}$, then the probability sum of all unscored derivation is small: $\sum_{g \in G^+} p(g) \leq \epsilon$.

To guarantee that $p_\theta(g) \leq \frac{\epsilon}{|G^+|}$, we *unroll* g until this stopping condition is satisfied. Unrolling a stream from $g = [d_1, d_2, \dots]$ means popping d_1 from g, constructing a singleton derivation stream $g_{\text{new}} = [d_1]$, pushing g_{new} to the agenda and scoring the remaining stream based on the next derivation $s(g) = s(d_2)$ (Figure 7).

To check if $p(g) \leq \frac{\epsilon}{|G^+|}$, we define the following upper bound U on $p(g)$, which is based on the number of derivations in the stream $|g|$:

$$p_\theta(g) = \frac{\sum_{d \in g} e^{s(d)}}{\sum_{g' \in G} \sum_{d' \in g'} e^{s(d')}} \leq \frac{|g| e^{s(g[1])}}{\sum_{g' \in G} e^{s(g'[1])}} = U$$

where $g[1]$ is the first derivation in g. Checking that $U \leq \frac{\epsilon}{|G^+|}$ is easy, since it is based only on the first derivation of every stream. Once all streams meet this criterion, we know that the total unscored probability is less than ϵ. As learning progresses, there are be many low probability derivations which we can skip entirely.

The last missing piece is ensuring that streams are sorted without explicitly scoring all derivations. We make a best effort to preserve this property.

Sorting derivation streams. All derivations in a stream g have the same child derivations, as they were constructed by one application of a semantic function f. Thus, the difference in their scores is only due to new features created when applying f. We can decompose these new features into two disjoint feature sets. One set includes features that depend on the grammar rule only and are independent of the input utterance x, and another also depends on x. For example, the semantic function $f = \text{LEX}$ maps phrases, such as *"born in"*, to logical forms, such as `PlaceOfBirthOf`. Most features extracted by LEX do not depend on x: the conjunction of *"born in"* and `PlaceOfBirthOf`, the frequency of the phrase *"born in"* in a corpus, etc. However, some features may depend on x as well. For example, if x is *"what city was abraham lincoln born in"*, we can conjoin `PlaceOfBirthOf` with the first two words *"what city"*. As another example, the semantic function BRIDGE takes unary predicates, such as `AbeLincoln`, and joins them with any type-compatible binary to produce logical forms, such as `PlaceOfBirthOf.AbeLincoln`. Again, a feature such as the number of assertions in $\mathcal{K}$ that contain `PlaceOfBirthOf` does not depend on x, while a feature that conjoins the introduced binary (`PlaceOfBirthOf`) with the main verb (*"born"*), does depend on x (see Section 8).

Our strategy is to pre-compute all features that are independent of x before training,[5] and sort streams

[5]For LEX, this requires going over all lexicon entries once. For BRIDGE, this requires going once over the KB.

based on these features only, as an approximation for the true order. Let's assume that derivations returned by an application of a semantic function f are parameterized by an auxiliary set $\mathcal{B}$. For example, when applying LEX on *"born in"*, $\mathcal{B}$ will include all lexical entries that map *"born in"* to a binary predicate. When applying BRIDGE on `AbeLincoln`, $\mathcal{B}$ will include all binary predicates that are type-compatible with `AbeLincoln`. We equip each $b \in \mathcal{B}$ with a feature vector $\phi_{\mathcal{B}}(b)$ (computed before training) of all features that are independent of x. This gives rise to a score $s_{\mathcal{B}}(b) = \phi_{\mathcal{B}}(b)^\top \theta$ that depends on the semantic function only. Thus, we can sort $\mathcal{B}$ before parsing, so that when the function f is called, we do not need to instantiate the derivations.

Note that the parameters θ and thus $s_{\mathcal{B}}$ change during learning, so we re-sort $\mathcal{B}$ after every iteration (of going through all training examples), yielding an approximation to the true ordering of $\mathcal{B}$. In practice, features extracted by LEX depend mostly on the lexical entry itself and our approximation is accurate, while for BRIDGE some features depend on x, as we explain next.

8 Features

The feature set in our model includes all features described in Berant et al. (2013).[6] In addition, we add new lexicalized features that connect natural language phrases to binary predicates.

In Berant et al. (2013), a binary predicate is generated using a lexicon constructed offline via alignment, or through the bridging operation. As mentioned above, bridging allows us to join unary predicates with binary predicates that are type-compatible, even when no word in the utterance triggers the binary predicate. For example, given the utterance *"what money to take to sri lanka"*, the parser will identify the entity `SriLanka`, and bridging will propose all possible binaries, including `Currency`. We add a feature template that conjoins binaries suggested by bridging (`Currency`) with all content word lemmas (*"what"*, *"money"*, *"take"*). After observing enough examples, we expect the feature corresponding to *"money"* and `Currency` to be upweighted. Generating freely and reweighting using features can be viewed as a soft way to expand the lexicon during training, similar to lexicon generation (Zettlemoyer and Collins, 2005). Note that this feature depends on the utterance x, and is not used for sorting streams (Section 7).

Finally, each feature is actually duplicated: one copy fires when choosing derivations on the agenda (Algorithm 1, line 4), and the other copy fires when choosing the final root derivation (line 6). We found that the increased expressivity from separating features improves accuracy.

9 Experiments

We evaluate our semantic parser on the WEBQUESTIONS dataset (Berant et al., 2013), which contains 5,810 question-answer pairs. The questions are about popular topics (e.g., *"what movies does taylor lautner play in?"*) and answers are sets of entities obtained through crowdsourcing (all questions are answerable by Freebase). We use the provided train-test split and perform three random 80%-20% splits of the training data for development.

We perform lexical lookup for Freebase entities using the Freebase Search API and obtain 20 candidate entities for every named entity identified by Stanford CoreNLP (Manning et al., 2014). We use the lexicon released by Berant et al. (2013) to retrieve unary and binary predicates. We execute λ-DCS logical forms by converting them to SPARQL and querying our local Virtuoso-backed copy of Freebase. During training, we use L_1 regularization, and crudely tune hyperparameters on the development set (beam size $K = 200$, tolerance for the lazy agenda $\epsilon = 0.01$, local reweighting $\beta = 1000$, and L_1 regularization strength $\lambda = 10^{-5}$).

We evaluated our semantic parser using the reward of the predictions, i.e., average F_1 score on predicted vs. true entities over all test examples.[7]

9.1 Main Results

Table 1 provides our key result comparing the fixed-order parser (FIXEDORDER) and our proposed agenda-based parser (AGENDAIL). In all subsequent tables, **Train**, **Dev.**, and **Test** denote training, development and test accuracies, |**Act.**| denotes

[6] As in previous work, some features use the fact that the spelling of KB predicates is often similar to English words.

[7] We use the official evaluation script from `http://www-nlp.stanford.edu/software/sempre/`.

	Test	Train	|Act.|	|Feat.|	Time
FIXEDORDER	49.6	60.6	18,127	18,127	1,782
AGENDAIL	**49.7**	**61.1**	**1,346**	**1,814**	**291**

Table 1: Test set results for the standard fixed-order parser (FIXEDORDER) and our new agenda-based parser (AGENDAIL), which substantially reduces parsing time and the number of parsing actions at no cost to accuracy.

System	Authors	Acc.
YV14	Yao and Van-Durme (2014)	35.4
BCFL13	Berant et al. (2013)	35.7
BDZZ14	Bao et al. (2014)	37.5
BWC14	Bordes et al. (2014)	39.2
BL14	Berant and Liang (2014)	39.9
YDZR14	Yang et al. (2014)	41.3
BWC14+ BL14	Bordes et al. (2014)	41.8
WYWH14	Wang et al. (2014)	45.3
WYWH14	Wang et al. (2014)	45.3
YCHG15	Yih et al. (2015)	**52.5**
FIXEDORDER	this work	49.6
AGENDAIL	this work	**49.7**

Table 2: Results on the WEBQUESTIONS test set.

the average number of parsing actions (pops from agenda in AGENDAIL and derivations placed on chart in FIXEDORDER) per utterance, **|Feat.|** denotes the average number of featurized derivations per utterance, and **Time** is average parsing time in milliseconds.

We found that AGENDAIL is 6x faster than FIXEDORDER, performs 13x fewer parsing actions, and reduces the number of featurized derivations by an order of magnitude, without loss of accuracy.

Table 2 presents test set results of our systems, compared to recently published results. We note that most systems perform question answering without semantic parsing. Our fixed-order parser, FIXEDORDER, and agenda-based parser, AGENDAIL, obtain an accuracy of 49.6 and 49.7 respectively. This improves accuracy compared to all previous systems, except for a recently published semantic parser presented by Yih et al. (2015), whose accuracy is 52.5. We attribute our accuracy improvement compared to previous systems to the new features and changes to the model, as we discuss below.

BCFL13 also used a fixed-order parser, but obtained lower performance. The main differences between the systems are that (i) our model includes new features (Section 8) combined with L_1 regularization, (ii) we use the Freebase search API rather than string matching, and (iii) our grammar generates a larger space of derivations.

Algorithm	Dev.	|Act.|	|Feat.|	Time
AGENDAIL	48.0	**1,421**	1,912	214
FIXEDORDER	**49.1**	18,259	18,259	1,972
AGENDA	45.9	6,211	6,320	419
FIXED+AGENDA	47.1	6,281	6,615	775
$\alpha = 1000$	47.8	11,279	11,279	1,216
$\alpha = 100$	35.6	3,858	3,858	174
$\alpha = 10$	27.0	1,604	**1,604**	**78**
p_θ^{+w}	43.3	1,706	2,121	238
p_θ^{+c}	36.8	3,758	4,278	358
p_θ	1.2	12,302	15,524	1,497
-BINARYANDLEMMA	40.5	1,561	2,110	167

Table 3: Development set results for variants of AGENDAIL.

9.2 Analysis

To gain insight into our system components, we perform extensive experiments on the development set.

Comparison with fixed-order parsing. Figure 8 compares accuracy, speed at test time, and number of derivations for AGENDAIL and FIXEDORDER. For AGENDAIL, we show both the number of derivations popped from the agenda, as well as number of derivations scored, which is slightly higher due to scored derivations on the agenda. We observe that for small beam sizes, AGENDAIL substantially outperforms FIXEDORDER. This is since AGENDAIL exploits small beams more efficiently in intermediate parsing states. For large beams performance is similar. In terms of speed and number of derivations, we see that AGENDAIL is dramatically more efficient than FIXEDORDER: with beam size 200–400, it is roughly as efficient as FIXEDORDER with beam size 10–20. For the chosen beam size ($K = 200$), AGENDAIL is 9x faster than FIXEDORDER.

For $K = 1$, performance is poor for AGENDAIL and zero for FIXEDORDER. This highlights the inherent difficulty of mapping to logical forms compared to more shallow tasks, as maintaining just a single best derivation for each parsing state is not sufficient.

A common variant on beam parsing is to replace the fixed beam size K with a threshold α, and prune any derivation whose probability is at least α times smaller than the best derivation in that state (Zhang et al., 2010; Bodenstab et al., 2011). We implemented this baseline and compared it to AGENDAIL

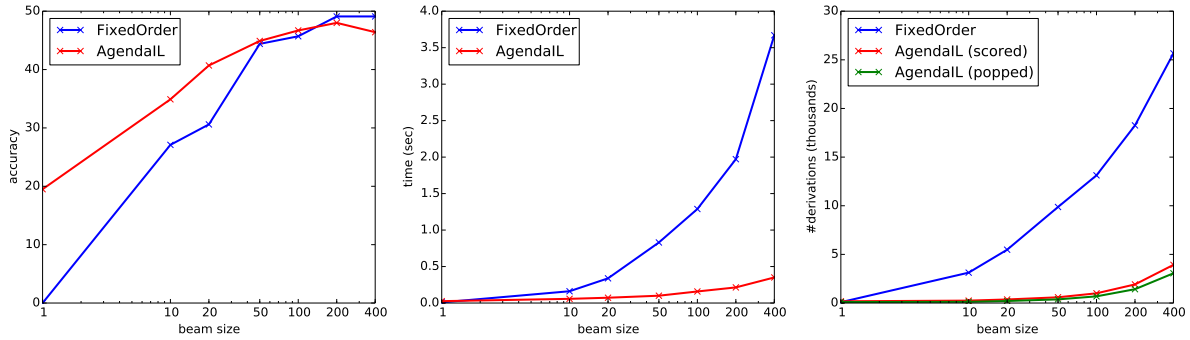

Figure 8: Comparing AGENDAIL and FIXEDORDER for various beam sizes (left: accuracy, middle: parsing time at test time in seconds, right: number of thousands of derivations scored and popped). The x-axis is on a logarithmic scale.

and FIXEDORDER in Table 3. We see that for $\alpha = 1000$, we get a faster algorithm, but a minor drop in performance compared to FIXEDORDER. However, this baseline still featurizes 6x more derivations and is 6x slower than AGENDAIL.

Impact of learning. The AGENDA baseline uses an agenda-based parser to approximate the gradients of (2). That is, we update parameters as in FIXEDORDER, but search for K root derivations using the agenda-based parser, described in Algorithm 1 (where we pop the highest scoring derivation). We observe that AGENDA featurizes 3x more derivations compared to AGENDAIL, and results in a 2.1 drop in accuracy. This demonstrates the importance of explicitly learning to choose correct actions during intermediate stages of parsing.

Since on the development set, FIXEDORDER outperformed AGENDAIL by 1.1 points, we implemented FIXED+AGENDA, where a fixed-order parser is used at training time, but an agenda-based parser is used at test time. This parser featurized 3.5x more derivations compared to AGENDAIL, is 3.5x slower, and has slightly lower accuracy.

Recall that AGENDAIL samples a history from $p_\theta^{+\text{cw}}$, that is, using local reweighting and history compression. Table 3 shows the impact of sampling from $p_\theta^{+\text{w}}$ (local reweighting), $p_\theta^{+\text{c}}$ (history compression), and directly from p_θ, which reduces to policy gradient. We observe that sampling from p_θ directly according to policy gradient results in very low accuracy, as this produces derivations with zero reward most of the time. Both local reweighting and history compression alone improve accuracy (local reweighting is more important), but both perform worse than AGENDAIL.

	Acc.		|Feat.|		Time	
	Tr.	**Dev.**	**Tr.**	**Dev.**	**Tr.**	**Dev.**
$\epsilon = 10^2$	56.4	46.7	**1,650**	2,121	**1,159**	345
$\epsilon = 10^{-1}$	59.5	47.0	2,043	1,890	1,425	279
$\epsilon = 10^{-2}$	61.0	48.0	2,600	1,912	1,830	**214**
$\epsilon = 10^{-3}$	**61.4**	**48.5**	3,063	**1,740**	2,110	220
NOSTREAM	60.0	47.6	4,049	4,931	3,155	293

Table 4: Accuracy, number of featurized derivations, and parsing time for both the training set and development set when varying the value of the tolerance parameter ϵ.

Impact of lazy agenda. We now examine the contribution of the lazy agenda. Note that the lazy agenda affects training time much more than test time for two reasons: (a) at test time we only need to pop the highest scoring derivation, and the overhead of a priority queue only grows logarithmically with the size of the agenda. During training, we need take a full pass over the agenda when sampling, and thus the number of items on the agenda is important; (b) at test time we never unroll derivation streams, only pop the highest scoring derivation (see Section 7).

In brief, using the lazy agenda results in a 1.5x speedup at training time. To understand the savings of the lazy agenda, we vary the value of the tolerance parameter ϵ. When ϵ is very high, we will never unroll derivation streams, because for all derivation streams $U \leq \frac{\epsilon}{|G^+|}$ (Section 7). This will be fast, but sampling could be inaccurate. As ϵ decreases, we unroll more derivations. We also compared to the NOSTREAM baseline, where the agenda holds derivations rather than derivation streams.

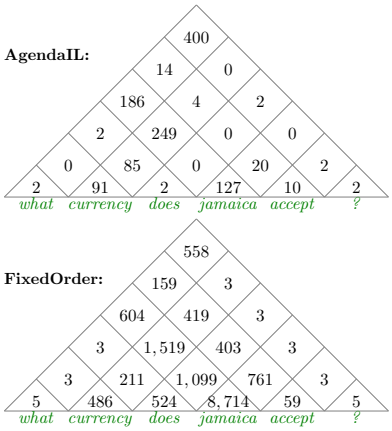

Figure 9: Number of derivations in every chart cell for the utterance *"what currency does jamaica accept?"*. AGENDAIL reduces the number of derivations in chart cells compared to FIXEDORDER.

Table 4 shows the results of these experiments. Naturally, the number of featurized derivations in training increases as ϵ decreases. In particular, NOSTREAM results in a 2.5x increase in number of featurized derivations compared to no unrolling ($\epsilon = 10^2$), and 1.5x increase compared to $\epsilon = 10^{-2}$, which is the chosen value. Similarly, average training time is about 1.5x slower for NOSTREAM compared to $\epsilon = 10^{-2}$.

Accuracy does not change much for various values of ϵ. Even when $\epsilon = 10^2$, accuracy decreases by only 1.8 points compared to $\epsilon = 10^{-3}$. Unexpectedly, NOSTREAM yields a slight drop in accuracy.

Feature ablation. Table 3 shows an ablation test on the new feature template we introduced that conjoins binaries and lemmas during bridging (-BINARYANDLEMMA). Removing this feature template substantially reduces accuracy compared to AGENDAIL, highlighting the importance of learning new lexical associations during training.

Example. As a final example, Figure 9 shows typical parse charts for AGENDAIL and FIXEDORDER. AGENDAIL generates only 1,198 derivations, while FIXEDORDER constructs 15,543 derivations, many of which are unnecessary.

In summary, we demonstrated that training an agenda-based parser to choose good parsing actions through imitation learning dramatically improves efficiency and speed at test time, while maintaining comparable accuracy.

10 Discussion and Related Work

Learning. In this paper, we sampled histories from a distribution that tries to target the reference derivation d^* whenever possible. Work in imitation learning (Abbeel and Ng, 2004; Daume et al., 2009; Ross et al., 2011; Goldberg and Nivre, 2013) has shown that interpolating with the model (corresponding to smaller β) can improve generalization. We were unable to improve accuracy by annealing β from 1000 to 0, so understanding this dynamic remains an open question.

Parsing. In this paper, we avoided computing K derivations in each chart cell using an agenda and learning a scoring function for choosing agenda items. A complementary and purely algorithmic solution is *lazy K-best parsing* (Huang and Chiang, 2005), or *cube growing* (Huang and Chiang, 2007), which do not involve learning or an agenda. Similar to our work, cube growing approximates the best derivations in each chart cell in the case where features do not decompose

Work in the past attempted to speed up inference using a simple model that is trained separately and used to prune the hypotheses considered by the main parsing model (Bodenstab et al., 2011; FitzGerald et al., 2013). We on the other hand speed up inference by training a single model that learns to follow good parsing actions.

Work in agenda-based syntactic parsing (Klein and Manning, 2003; Pauls and Klein, 2009) focused on A* algorithms where each derivation has a priority based on the derivation score (*inside score*), and a completion estimate (*outside score*). Good estimates for the outside score result in a decrease in the number of derivations. Currently actions depend on the inside score, but we could add features based on chart derivations to provide "outside" information. Adding such features would present computational challenges as scores on the agenda would have to be updated as the agenda and chart are modified.

Semantic parsing has been gaining momentum in recent years, but still there has been relatively little work on developing faster algorithms, especially compared to syntactic parsing (Huang, 2008; Kummerfeld et al., 2010; Rush and Petrov, 2012; Lewis

and Steedman, 2014). While we have obtained significant speedups, we hope to encourage new ideas that exploit the structure of semantic parsing to yield better algorithms.

Reproducibility. All code,[8] data, and experiments for this paper are available on the CodaLab platform at `https://www.codalab.org/worksheets/0x8fdfad310dd84b7baf683b520b4b64d5/`.

Acknowledgments

We thank the anonymous reviewers and the action editor, Jason Eisner, for their thorough reviews and constructive feedback. We also gratefully acknowledge the support of the DARPA Communicating with Computers (CwC) program under ARO prime contract no. W911NF-15-1-0462.

References

P. Abbeel and A. Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*.

M. Auli and A. Lopez. 2011. Efficient CCG parsing: A* versus adaptive supertagging. In *Association for Computational Linguistics (ACL)*.

J. Bao, N. Duan, M. Zhou, and T. Zhao. 2014. Knowledge-based question answering as machine translation. In *Association for Computational Linguistics (ACL)*.

J. Berant and P. Liang. 2014. Semantic parsing via paraphrasing. In *Association for Computational Linguistics (ACL)*.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

N. Bodenstab, A. Dunlop, K. Hall, and B. Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Association for Computational Linguistics (ACL)*, pages 440–449.

A. Bordes, S. Chopra, and J. Weston. 2014. Question answering with subgraph embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Q. Cai and A. Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Association for Computational Linguistics (ACL)*.

[8] Our system uses the SEMPRE toolkit (`http://nlp.stanford.edu/software/sempre`).

S. A. Caraballo and E. Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24:275–298.

K. Chang, A. Krishnamurthy, A. Agarwal, H. Daume, and J. Langford. 2015. Learning to search better than your teacher. *arXiv*.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Computational Natural Language Learning (CoNLL)*, pages 18–27.

H. Daume, J. Langford, and D. Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75:297–325.

J. Duchi, E. Hazan, and Y. Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*.

N. FitzGerald, Y. Artzi, and L. S. Zettlemoyer. 2013. Learning distributions over logical forms for referring expression generation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1914–1925.

Y. Goldberg and J. Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics (TACL)*, 1.

Google. 2013. Freebase data dumps (2013-06-09). `https://developers.google.com/freebase/data`.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64.

L. Huang and D. Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Association for Computational Linguistics (ACL)*.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Association for Computational Linguistics (ACL)*.

J. Jiang, A. Teichert, J. Eisner, and H. Daume. 2012. Learned prioritization for trading off accuracy and speed. In *Advances in Neural Information Processing Systems (NIPS)*.

M. Kay. 1986. *Algorithm Schemata and Data Structures in Syntactic Processing*. Readings in Natural Language Processing.

D. Klein and C. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Human Language Technology and North American Association for Computational Linguistics (HLT/NAACL)*.

J. Krishnamurthy and T. Mitchell. 2012. Weakly supervised training of semantic parsers. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 754–765.

J. Kummerfeld, J. Roesner, T. Dawborn, J. Haggerty, J. Curran, and S. Clark. 2010. Faster parsing by supertagger adaptation. In *Association for Computational Linguistics (ACL)*.

T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1223–1233.

T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing (EMNLP)*.

M. Lewis and M. Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Empirical Methods in Natural Language Processing (EMNLP)*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

P. Liang. 2013. Lambda dependency-based compositional semantics. *arXiv*.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL system demonstrations*.

C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. 2012. A joint model of language and perception for grounded attribute learning. In *International Conference on Machine Learning (ICML)*, pages 1671–1678.

A. Pauls and D. Klein. 2009. K-best A* parsing. In *Association for Computational Linguistics (ACL)*, pages 958–966.

S. Ross, G. Gordon, and A. Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*.

A. Rush and S. Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Human Language Technology and North American Association for Computational Linguistics (HLT/NAACL)*.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*.

S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Z. Wang, S. Yan, H. Wang, and X. Huang. 2014. An overview of Microsoft deep QA system on Stanford WebQuestions benchmark. Technical report, Microsoft Research.

Y. W. Wong and R. J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Association for Computational Linguistics (ACL)*, pages 960–967.

M. Yang, N. Duan, M. Zhou, and H. Rim. 2014. Joint relational embeddings for knowledge-based question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*.

X. Yao and B. Van-Durme. 2014. Information extraction over structured data: Question answering with Freebase. In *Association for Computational Linguistics (ACL)*.

W. Yih, M. Chang, X. He, and J. Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Association for Computational Linguistics (ACL)*.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 678–687.

Y. Zhang, B. Ahn, S. Clark, C. V. Wyk, J. R. Curran, and L. Rimell. 2010. Chart pruning for fast lexicalised-grammar parsing. In *International Conference on Computational Linguistics (COLING)*.