Learned Prioritization for Trading Off Accuracy and Speed

Jiarong Jiang[*] JIARONG@UMIACS.UMD.EDU
Adam Teichert[†] TEICHERT@JHU.EDU
Hal Daumé III[*] HAL@UMIACS.UMD.EDU
Jason Eisner[†] JASON@CS.JHU.EDU

[*]University of Maryland, College Park, Maryland 20783
[†]Johns Hopkins University, Baltimore, Maryland 21218

Abstract

Users want natural language processing (NLP) systems to be both fast and accurate, but quality often comes at the cost of speed. The field has been manually exploring various speed-accuracy tradeoffs for particular problems or datasets. We aim to explore this space automatically, focusing here on the case of agenda-based syntactic parsing (Kay, 1986). Unfortunately, off-the-shelf reinforcement learning techniques fail to learn good policies: the state space is too large to explore naively. We propose a hybrid reinforcement/apprenticeship learning algorithm that, even with few inexpensive features, can automatically learn weights that achieve competitive accuracies at significant improvements in speed over state-of-the-art baselines.

1. Introduction

The nominal goal of natural language processing (NLP) is to achieve high accuracy. Unfortunately, high accuracy often comes at the price of slow computation.

Typically one seeks a "reasonable" trade-off between accuracy and speed. But what is "reasonable" for one person might not be reasonable for another. Our goal is to optimize a system with respect to a user-specified speed/accuracy trade-off, on a user-specified data distribution. We formalize our problem in terms of learning priority functions for generic inference algorithms (Section 2).

Much research has been dedicated to finding exact or approximate speedups in a wide range of inference problems including sequence tagging (Kaji et al., 2010), constituent parsing (Pauls & Klein, 2009; Finkel et al., 2008), dependency parsing (Goldberg & Elhadad, 2010), and machine translation (Petrov et al., 2008). Many of the speedup strategies in the literature can be expressed as pruning or prioritization heuristics. Prioritization heuristics govern the order in which search actions are taken while pruning heuristics explicitly dictate whether particular actions should be taken at all. Examples of prioritization include A* (Klein & Manning, 2003; Haghighi et al., 2007) and Hierarchical A* (Pauls & Klein, 2010) heuristics, which, in the case of agenda-based parsing, prioritize parse actions so as to reduce work while maintaining the guarantee that the most likely parse is found. Alternatively, classifier-based pruning (Roark & Hollingshead, 2008), beam-width prediction (Bodenstab et al., 2011), coarse-to-fine pruning (Charniak et al., 2006; Petrov & Klein, 2007) and figure-of-merit prioritization (Caraballo & Charniak, 1998; Charniak et al., 1998) can result in even faster inference if a small amount of search error can be tolerated.

Unfortunately, deciding which techniques to use for a specific setting can be difficult: it is impractical to "try everything." In this paper, by combining reinforcement learning and apprenticeship learning techniques, we develop a learning algorithm that can successfully learn such a trade-off in the context of constituency parsing. Although this paper focuses on parsing, we expect the approach to transfer to prioritization under other agenda-based inference algorithms, such as in machine translation.

2. Priority-based Inference

Inference algorithms in NLP (e.g. parsers, taggers, or translation systems) as well as more broadly in artificial intelligence (e.g., planners) often rely on prioritized exploration. For concreteness, we describe inference in the context of parsing, though it is well known that this setting captures all the essential structure of a much larger family of "deductive inference" problems (Kay, 1986; Ramakrishnan, 1991; Sikkel, 1997; Shieber et al., 1995; Goodman, 1999;

Presented at the International Conference on Machine Learning (ICML) workshop on *Inferning: Interactions between Inference and Learning*, Edinburgh, Scotland, UK, 2012.

Eisner et al., 2005; Eisner & Blatz, 2007).

2.1. Prioritized Parsing

Given a grammar, perhaps the simplest approach to inferring the best parse tree for a given sentence is to assemble the parse from the bottom up as in CKY (Younger, 1967). A standard extension of the CKY algorithm uses an "agenda"—a priority queue of constituents built so far—to decide what to do next (Kay, 1986).

Our goal is to *learn* a prioritization function that is able to prioritize "good" actions (a high accuracy and fast solution) before "bad" actions. In order to operationalize this approach, we need to define the test-time objective function we wish to optimize; we choose a simple linear interpolation of accuracy and speed:

$$\text{quality} = \text{accuracy} - \lambda \times \text{time} \tag{1}$$

where we can choose an λ that reflects our true preferences, or investigate different values of λ to obtain a speed and accuracy tradeoff. We halt inference as soon as the parser pops its first complete parse and formulate this problem as a Markov Decision Process (MDP).

2.2. Inference as a Markov Decision Process

A Markov Decision Process (MDP) is a formalization of a memoryless search process. An MDP consists of a *state space* S, an *action space* A, and a *transition function* T. For parsing, the state is the full current chart and agenda (and is astronomically large: roughly 10^{17} states for average sentences). The agent controls which item (constituent) to "pop" from the agenda. Its possible actions correspond to items currently on the agenda. When the agent pops item y, the environment deterministically adds y to the chart, combines y as licensed by the grammar with adjacent items z in the chart, and places each resulting new item x on the agenda. (Duplicates are merged: the highest probability one is kept.) The only stochasticity is the initial draw of a new sentence to be parsed.

We are interested in learning a deterministic policy that always pops the highest-priority available action. Thus, learning a policy corresponds to learning a priority function. We define the priority of action a in state s as the dot product of a feature vector $\boldsymbol{\phi}(a, s)$ with the weight vector $\boldsymbol{\theta}$. Formally, our policy is

$$\pi_{\boldsymbol{\theta}}(s) = \arg\max_a \ \boldsymbol{\theta} \cdot \boldsymbol{\phi}(a, s) \tag{2}$$

2.3. Features for Prioritized Parsing

We use a very simple set of features: (1) Viterbi inside score;
(2) Does constituent touch the start of sentence;
(3) Does constituent touch the end sentence;
(4) Length of the constituent;
(5) Ratio of constituent length to sentence length;
(6) $\log p(\text{constituent label} \mid \text{prev. word POS tag})$ and $\log p(\text{constituent label} \mid \text{next word POS tag})$, where the POS tag of w is taken to be $\arg\max_t p(w \mid t)$ under the grammar;
(7) Case pattern of first word in constituent and previous/next word: upper case, lower case, number and symbol;
(8) Punctuation pattern in constituent: such as word, $ word, word $ word, etc. We use the five most indicative features for a span not being a constituent.
The log-probability features (1) and (6) are inspired by work on figures of merit for agenda-based parsing (Caraballo & Charniak, 1998); while the case and punctuation patterns (7) and (8) are inspired by structure-free parsing (Liang et al., 2008).

3. Reinforcement Learning

Reinforcement learning (RL) provides a generic solution to solving learning problems with delayed reward. In general the reward function may be stochastic, but in our case, it is deterministic: $r(s, a) \in \mathbb{R}$. The reward function we consider is: $r(s, a) =$

$$\begin{cases} \text{acc}(a) - \lambda \cdot \text{time}(s) & \text{if } a \text{ is a full parse tree} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Here, $\text{acc}(a)$ measures the accuracy of the full parse tree popped by the action a (against a gold standard) and $\text{time}(s)$ is a user-defined measure of time.

3.1. Boltzmann Exploration

At test time, the transition between states is deterministic: our policy always chooses the action a that has highest priority in the current state s. However, during training, we promote exploration of policy space by running with stochastic policies: $\pi_{\boldsymbol{\theta}}(a \mid s)$. Thus, there is some chance of popping a lower-priority action, to find out if it is useful and should be given higher-priority. In particular, we use Boltzmann exploration to construct a stochastic policy with a Gibbs distribution. Our policy is:

$$\pi_{\boldsymbol{\theta}}(a \mid s) = \frac{1}{Z(s)} \exp\left[\frac{1}{\textit{temp}}\, \boldsymbol{\theta} \cdot \boldsymbol{\phi}(a, s)\right] \tag{4}$$

That is, the log-likelihood of action a at state s is an affine function of its priority. The temperature *temp* controls the amount of exploration. As $\textit{temp} \to 0$, π approaches the deterministic policy in Eq (2); as $\textit{temp} \to \infty$, π approaches the uniform distribution over available actions. During training, *temp* can be decreased to shift from exploration to exploitation.

As we have a fixed-horizon episodic task (i.e., the agent does not live forever), a stochastic policy π gives rise to a separate total reward R for each episode, i.e., each time the parser parses a sentence. We want our policy to maximize the *expected total reward*, which is fully defined by π together with the distribution over the random behavior of the environment (transitions and rewards). We can measure this by simulation.

Note that even in our deterministic setting, the environment does present a little bit of randomness: at the start of each episode, it chooses a random sentence to be parsed. At training time, these sentences come from the training set, and at test time they come from the test set.

A trajectory τ is the complete sequence of state/action/reward triples from parsing a single sentence (an "episode"). As is common, we denote $\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T \rangle$, where: s_0 is the starting state; a_t is chosen by the agent by $\pi_{\boldsymbol{\theta}}(a_t \mid s_t)$; $r_t = r(s_t, a_t)$; and s_{t+1} is drawn by the environment from $T(s_{t+1} \mid s_t, a_t)$, deterministically in our case. At a given temperature, the weight vector $\boldsymbol{\theta}$ gives rise to a distribution over trajectories through the state space and hence to an expected reward:

$$R = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} r_t \right]. \tag{5}$$

3.2. Policy Gradient

We wish to find parameters that yield the highest possible expected reward. We carry out this optimization using a stochastic gradient ascent algorithm known as policy gradient (Baxter & Bartlett, 2001; Williams, 1992; Sutton et al., 2000). This operates by taking derivatives of the reward with respect to $\boldsymbol{\theta}$ and taking steps in that direction:

$$\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) = p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \tag{6}$$

so the gradient of the objective becomes

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau}[R(\tau)] = \mathbb{E}_{\tau}\Big[R(\tau) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t) \Big] \tag{7}$$

Here τ is a random trajectory chosen by policy $\pi_{\boldsymbol{\theta}}$, and r_t is the reward at step t of τ. The expectation can be approximated by sampling trajectories. It also requires computing the gradient of each policy decision, which, by Eq (4), is: $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) =$

$$\frac{1}{\mathit{temp}} \left(\boldsymbol{\phi}(a_t, s_t) - \sum_{a' \in A} \pi_{\boldsymbol{\theta}}(a' \mid s_t) \boldsymbol{\phi}(a', s_t) \right) \tag{8}$$

Combining Eq (7) and Eq (8) gives the form of the gradient with respect to a single trajectory. The policy gradient algorithm samples one (or several) trajectory according to the current $\pi_{\boldsymbol{\theta}}$, and then takes a gradient step according to Eq (7). This increases the probability of actions on high-reward trajectories more than actions on low-reward trajectories.

The main difficulty with policy gradient is that it has no way to determine which actions were "responsible" for a trajectory's reward. Without causal reasoning, we need to sample many trajectories in order to distinguish which actions are reliably associated with higher-reward. This is a significant problem for us, since the average trajectory length of an A* parser on a 15 word sentence is about $30k$ steps, only about 40 of which (less than 0.15%) are actually needed to successfully complete the parse optimally. This is the *credit assignment problem.*

3.3. Reward Shaping

A classic approach to attenuating the credit assignment problem when one has some knowledge about the domain is *reward shaping* (Gullapalli & Barto, 1992; Matarić, 1994). The goal of reward shaping is to give the agent reward *earlier* in a trajectory in order to improve its convergence rate. Under suitable conditions (essentially that the reward can be written as the difference of two potential functions), one can prove that reward shaping preserves the *optimal* behavior of an agent on an MDP with a fixed reward function, while reducing the time used attempting suboptimal actions (Ng et al., 1999).

Our shaping approach is to distribute the fully delayed reward in accordance with our prior beliefs about the contribution of the action to the final reward. If speed is measured by the number of popped items and accuracy is measured by labeled constituent recall of the first popped full tree, we can shape the reward as:

$$\tilde{r}(s, a) = \begin{cases} \delta(a) - \lambda & \text{if } a \text{ is a full parse tree} \\ 1 - \lambda & \text{if } a \text{ is in the true parse} \\ -\lambda & \text{otherwise} \end{cases} \tag{9}$$

Here, $\delta(s)$ is a negative reward assigned to compensate for actions which received early reward for constituents that did not end up in the final parse. It is easy to show that for a full trajectory that ends in a complete parse tree, this reward coincides with the unshaped reward from Eq (3): $r(\tau) = \tilde{r}(\tau)$.

4. Apprenticeship Learning

To help guide the learning process, we would like to explore more intelligently than Boltzmann exploration, in particular, focusing on high-reward regions of policy space. We introduce *oracle actions* as a guidance. At the beginning of the learning process, the parser has little knowledge of what constituents are useful to build (i.e. which action-

s would tend to improve accuracy enough to be worth the cost in speed). The oracle actions can give a good idea of reasonable things to do and what subset of the enormous search space to explore.

An oracle action should be one that leads to a maximum-reward tree, where reward is defined in terms of accuracy *and* speed (Eq (3)). However, several oracle actions may be available at once. First, our agenda-based algorithm has some freedom in how it orders its actions: e.g., it can arrive at the optimal tree by building either the subject noun-phrase first or the main verb-phrase first. Second, there may be multiple optimal trees with equal reward: e.g., for state-of-the-art grammars like the ones we use (Petrov & Klein, 2008), the grammar produces fine-grained parses, but accuracy is only measured against the coarse-grained parses provided in labeled data. Thus, two fine-grained parses are equally accurate if they disagree only on latent variables.

There are many different ways to get oracle trees. For simplicity, in the experiments, we use the ground truth of a sentence as the oracle tree (Details in experiment section). Alternatively, if the full set of possible parses of a sentence is accessible, we can find out the exact parse with the best speed-accuracy tradeoff and use that as the oracle tree. Or, we can run the agenda-based parser but with the pushed-back reward as the priority to get the oracle tree. However, these initial choices should not influence the results at the end of reinforcement learning if we gradually increase the ability of stochastic exploration for the parser.

We address this problem by letting the current policy decide which of the many possible oracle actions it likes the bestWe let our oracle actions be those that build constituents in the gold parse tree since this tends to result in both fast and accurate parsing in most cases.

4.1. Apprenticeship Learning via Classification

Given access to a shaped reward that assigns a cost at each step (Eq (9)) *and* a notion of oracle actions, a straightforward approach is to train a classifier to make search decisions, a popular approach in incremental parsing (Ratnaparkhi, 1999; Collins & Roark, 2004; Charniak, 2010), and the initial iteration of the state-of-the-art apprenticeship learning algorithm, DAGGER (Ross et al., 2011).

We train a classifier as follows. Trajectories are generated by following oracle actions. At each step in the trajectory, (s_t, a_t), a classification example is generated, where the action taken by the oracle (a_t) is considered the correct class and all other available actions are incorrect. The classifier that we train on these examples is a maximum entropy classifier. In fact, the *gradient* of this classifier is nearly identical to the gradient of policy gradient (Eq (7)) except that the *total reward* $R(\tau)$ is replaced by the immediate reward r_t, which is constant for all oracle steps.

An obvious practical issue with the classification-based approach is that it trains the classifier only at states visited by the oracle. This leads to the well-known problem that it is unable to learn to recover from past errors (Bagnell, 2005; Daumé III & Marcu, 2005; Xu & Fern, 2007; Xu et al., 2007; Ratliff et al., 2009; Daumé III et al., 2009; Ross et al., 2011). An additional issue is that not all errors are created equal. Some incorrect actions are more expensive than others, if they create constituents that can get combined in many locally-attractive ways. But our classifier does not know this. A final issue has to do with the nature of the oracle.Our oracle action selector *ignores* the trade-off between accuracy and speed, and only focuses on accuracy.

5. Oracle-Infused Policy Gradient

The failure of both standard reinforcement learning algorithms and standard apprenticeship learning algorithms on our problem leads us to develop a new approach. To achieve this, we define the notion of an oracle-infused policy. Let π be an arbitrary policy and let $\delta \in [0, 1]$. We define the oracle infused policy π^+_δ as follows:

$$\pi^+_\delta(a \mid s) = \delta\pi^*(a \mid s) + (1 - \delta)\pi(a \mid s) \qquad (10)$$

In other words, when choosing an action, π^+_δ explores the policy space with probability $1 - \delta$, and with probability δ, we force it to take an oracle action.

Our algorithm proceeds by taking policy gradient steps with respect to trajectories drawn from π^+_δ rather than π. Because of this, it is theoretically important to ensure that in the limit of learning, $\delta \to 0$ so that trajectories are being drawn from the actual learned policy. We use $\delta = 0.8^{\text{epoch}}$, where epoch is the total number of passes made through the training set at that point. Note that if $\delta = 1$, this reduces to the classifier-based approach; and if $\delta = 0$, this reduces to policy gradient. By selecting delayed reward and $\delta = 1$, this can be regarded as an approximation to Searn that uses random roll-outs rather than "all possible" roll-outs.

6. Experiments

All of our experiments are based on the Wall Street Journal portion of the Penn Treebank (Marcus et al., 1993). We use a latent-variable context-free grammar for parsing (Petrov & Klein, 2007). In the model selection, we estimate this grammar using 5 split-merge iterations on sections 2–20 of the Treebank, reserving section 22 for learning the parameters of our policy. In particular, for Section 6.1, the same 100 sentences of at most 15 words from section 22 were used for training and test. We measure accuracy in terms of labeled recall (including preterminals) and measure speed in terms of the number of pops/pushes performed on the a-

Model	# of pops	Recall	F1
D−	686641	56.35	58.74
I−	187403	76.48	76.92
D+	1275292	84.17	83.38
I+	682540	91.16	91.33
UC (no pruning)	1496080	93.34	93.19

Table 1. Performance on 100 sentences with $\lambda = 10^{-6}$.

genda. The limitation to relatively short sentences is purely for improved efficiency at training time.

Our baselines are: **(HA*)** a Hierarchical A* parser (Pauls & Klein, 2009) with same pruning threshold at each hierarchy level; **(UC)** an A* parser with a 0 heuristic function and pruning; **(A$^*_p$)** a **UC** variant, on which we decrease the pruning threshold if no tree is returned;[1] and **(CTF)** an agenda-based coarse-to-fine parser (Petrov & Klein, 2007).

6.1. Learned Prioritization Approaches

We specifically evaluate four variants of our learned prioritization approach:

D−	delayed reward in a classifier	(sec. 3.2)
I−	immediate reward in a classifier	(sec. 3.3)
D+	delayed reward with oracle-infusion	(sec. 5)
I+	immediate reward with oracle-infusion	(sec. 5)

Table 1 shows the result on the 100 sentences. We can see that the classifier-based approaches perform poorly. When the training trajectory consists of only oracle actions, the we suffer from a biased set of training trajectories; hence, the model cannot learn useful test-time weights. On the other hand, without any help from the oracle actions, we suffer from large variance in training trajectories: it takes days to train the parser and training does not converge. Our "oracle-infused" compromise uses *some* oracle actions, and after several passes through the data, the parser learns to make good decisions without help from the oracle.

6.2. Pareto Frontier

Our final evaluation is on the held-out test set (length-limited sentences from Section 23). A 5-split grammar trained on section 2-21 is used. Given our previous results in Table 1, we only consider the **I+** model. To investigate different points of speed-accuracy tradeoff, we learn and then evaluate a policy for each of several settings of λ. We train our policy using sentences of at most 15 words from Section 22. We measured accuracy in recall points while we evaluate speed in terms of the number of pops/pushes performed on the agenda.

[1]This forces the parser to return *some* tree, just like our parser. When the **UC** parser with heavy pruning fails to find a tree, we reparse at the next lighter pruning level, similar to iterative-deepening search (Russell & Norvig, 1995). The runtime is assessed *cumulatively*.

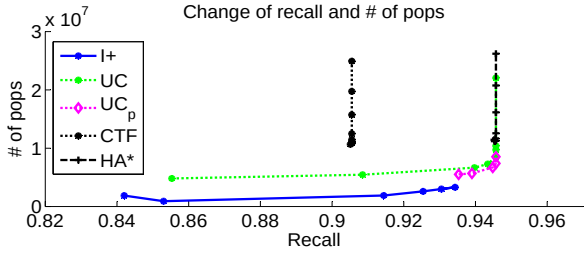

Figure 1. Pareto frontiers: Our **I+** parser at different values of λ, against the baselines at different pruning levels.

Figure 1 shows the baselines at different pruning thresholds as well as the performance of our policies trained using **I+** for $\lambda \in \{10^{-3}, 10^{-4}, \ldots, 10^{-8}\}$, using agenda pops as the measure of time. **I+** is 3 times as fast as **UC** at the cost of about 1% drop in accuracy (F-score from 94.58 to 93.56). **I+** can achieve the same accuracy as the pruned version of **UC** while still being twice as fast. **I+** also improves upon **HA*** and **UC$_p$** with respect to speed and eliminates at 60% of the pops. **I+** always does better than the coarse-to-fine parser (**CTF**) in terms of **both** speed and accuracy, though using the number of agenda pops as our measure of speed puts both hierarchical methods at a disadvantage.

We also ran experiments using the number of agenda pushes as the measure of time, again sweeping over settings of λ. With 1% recall (F-score from 94.58 to 93.54), it is at least four times as fast as **UC** (pushes from around 8 billion to 2 billion). While pruning speeds up **UC** significantly, we can also employ the same pruning in **I+**.

One limitation of our current model is that our policy explicitly returns the first tree that is found. This makes it hard to learn good weights when λ is too small. As $\lambda \to 0$, we should learn to simply parse exhaustively by not return a parse until the chart has been entirely filled in.

7. Conclusions and Future Work

In this paper, we considered the application of both reinforcement learning and apprenticeship learning to prioritize search in a way that is sensitive to a user-defined trade-off between speed and accuracy. We found that a novel oracle-infused variant of the policy gradient algorithm for reinforcement learning is effective for learning a fast and accurate parser with only a simple set of features. In addition, we uncovered many properties of this problem that separate it from more standard learning scenarios, and designed experiments to determine the *reasons* off-the-shelf learning algorithms fail.

An important avenue for future work is to consider richer feature sets, including "dynamic" features that depend on both the action *and* the state of the chart and agenda.They can also be used to decide when to halt.

References

Abbeel, Pieter and Ng, Andrew. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. In *Robotics and Autonomous Systems*, 2009.

Bagnell, J. Andrew. Robust supervised learning. In *AAAI*, 2005.

Baxter, J. and Bartlett, P. Infinite-horizon policy-gradient estimation. *JAIR*, 15:319–350, 2001.

Bodenstab, Nathan, Dunlop, Aaron, Hall, Keith, and Roark, Brian. Beam-width prediction for efficient CYK parsing. In *ACL*, 2011.

Boyd, S., Ghaoui, L. El, Feron, E., and Balakrishnan, V. Linear matrix inequalities in system and control theory. *SIAM*, 15, 1994.

Branavan, S.R.K., Chen, Harr, Zettlemoyer, Luke, and Barzilay, Regina. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.

Caraballo, Sharon A. and Charniak, Eugene. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, 1998. URL http://www.cs.brown.edu/people/sc/NewFiguresofMerit.ps.Z.

Charniak, E., Goldwater, S., and Johnson, M. Edge-based best-first chart parsing. In *Proceedings of the Workshop on Very Large Corpora*, pp. 127–133, 1998.

Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., et al. Multilevel coarse-to-fine PCFG parsing. In *NAACL/HLT*, pp. 168–175. Association for Computational Linguistics, 2006.

Charniak, Eugene. Top-down nearly-context-sensitive parsing. In *EMNLP*, 2010.

Collins, Michael and Roark, Brian. Incremental parsing with the perceptron algorithm. In *ACL*, 2004.

Daumé III, Hal and Marcu, Daniel. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

Daumé III, Hal, Langford, John, and Marcu, Daniel. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009. ISSN 0885-6125.

Eisner, Jason and Blatz, John. Program transformations for optimization of parsing algorithms and other weighted logic programs. In Wintner, Shuly (ed.), *Proceedings of the Conference on Formal Grammar (FG)*, pp. 45–85. CSLI Publications, 2007. URL http://cs.jhu.edu/~jason/papers/#fg06.

Eisner, Jason, Goldlust, Eric, and Smith, Noah A. Compiling comp ling: Weighted dynamic programming and the Dyna language. In *HLT/EMNLP*, pp. 281–290, Vancouver, October 2005. Association for Computational Linguistics. URL http://cs.jhu.edu/~jason/papers/#emnlp05-dyna.

Finkel, J.R., Kleeman, A., and Manning, C.D. Efficient, feature-based, conditional random field parsing. In *ACL*, pp. 959–967, 2008.

Goldberg, Y. and Elhadad, M. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL/HLT*, pp. 742–750. Association for Computational Linguistics, 2010. ISBN 1932432655.

Goodman, Joshua. Semiring parsing. *Computational Linguistics*, 25(4):573–605, December 1999. URL http://research.microsoft.com/~joshuago/finalring.ps.

Gullapalli, V. and Barto, A. G. Shaping as a method for accelerating reinforcement learning. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 1992.

Haghighi, Aria, DeNero, John, and Klein, Dan. Approximate factoring for A* search. In *NAACL/HLT*, pp. 412–419, Rochester, New York, April 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N07/N07-1052.

Kaji, N., Fujiwara, Y., Yoshinaga, N., and Kitsuregawa, M. Efficient staggered decoding for sequence labeling. In *ACL*, pp. 485–494. Association for Computational Linguistics, 2010.

Kalman, R. Contributions to the theory of optimal control. *Bol. Soc. Mat. Mexicana*, 5:558–563, 1968.

Kay, Martin. Algorithm schemata and data structures in syntactic processing. In Grosz, B. J., Sparck Jones, K., and Webber, B. L. (eds.), *Readings in Natural Language Processing*, pp. 35–70. Kaufmann, 1986. First published (1980) as Xerox PARC TR CSL-80-12.

Klein, Dan and Manning, Chris. Parsing and hypergraphs. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, 2001.

Klein, Dan and Manning, Chris. A* parsing: Fast exact Viterbi parse selection. In *NAACL/HLT*, 2003.

Liang, Percy, Daumé III, Hal, and Klein, Dan. Structure compilation: Trading structure for features. In *ICML*, Helsinki, Finland, 2008.

Marcus, M.P., Marcinkiewicz, M.A., and Santorini, B. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):330, 1993. ISSN 0891-2017.

Mataric, M. J. Reward functions for accelerated learning. In *ICML*, 1994.

Natarajan, S., Joshi, S., Tadepalli, P., Kersting, K., and Shavlik, J. Imitation learning in relational domains: A functional-gradient boosting approach. In *IJCAI*, 2011.

Nederhof, Mark-Jan. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143, 2003. doi: http://dx.doi.org/10.1162/089120103321337467.

Neu, G. and Szepesvári, Cs. Training parsers by inverse reinforcement learning. *Machine Learning*, 77, 2009.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

Ng, Andrew and Russell, Stuart. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

Pauls, A. and Klein, D. Hierarchical search for parsing. In *NAACL/HLT*, pp. 557–565. Association for Computational Linguistics, 2009.

Pauls, A. and Klein, D. Hierarchical A* parsing with bridge outside scores. In *ACL*, pp. 348–352. Association for Computational Linguistics, 2010.

Petrov, S. and Klein, D. Improved inference for unlexicalized parsing. In *NAACL/HLT*, pp. 404–411, 2007.

Petrov, S., Haghighi, A., and Klein, D. Coarse-to-fine syntactic machine translation using language projections. In *EMNLP*, pp. 108–116. Association for Computational Linguistics, 2008.

Petrov, Slav and Klein, Dan. Sparse multi-scale grammars for discriminative latent variable parsing. In *EMNLP*, pp. 867–876, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1091.

Ramakrishnan, Raghu. Magic templates: a spellbinding approach to logic programs. *Journal of Logic Programming*, 11(3-4): 189–216, 1991. ISSN 0743-1066. doi: http://dx.doi.org/10.1016/0743-1066(91)90026-L.

Ratliff, Nathan, Bradley, David, Bagnell, J. Andrew, and Chestnutt, Joel. Boosting structured prediction for imitation learning. In *NIPS*, 2007.

Ratliff, Nathan, Silver, David, and Bagnell, J. Andrew. Learning to search: Functional gradient techniques for imitation learning. In Peters, Jan and Ng, Andrew Y. (eds.), *Autonomous Robots*, volume 27, pp. 25–53. Springer, July 2009.

Ratnaparkhi, A. Learning to parse natural language with maximum entropy models. *Machine learning*, 34(1):151–175, 1999.

Roark, Brian and Hollingshead, Kristy. Classifying chart cells for quadratic complexity context-free inference. In *COLING*, pp. 745–752, Manchester, UK, August 2008. Coling 2008 Organizing Committee. URL http://www.aclweb.org/anthology/C08-1094.

Ross, Stephane and Bagnell, J. Andrew. Efficient reductions for imitation learning. In *AI-Stats*, 2010.

Ross, Stephane, Gordon, Geoff J., and Bagnell, J. Andrew. A reduction of imitation learning and structured prediction to no-regret online learning. In *AI-Stats*, 2011.

Russell, Stuart and Norvig, Peter. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 1995.

Shieber, Stuart M., Schabes, Yves, and Pereira, Fernando. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, 1995. URL http://www.eecs.harvard.edu/~shieber/Biblio/Papers/infer.pdf.

Sikkel, Klaus. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Texts in Theoretical Computer Science. Springer-Verlag, 1997.

Sutton, Richard and Barto, Andrew. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Sutton, Richard S., McAllester, David, Singh, Satinder, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pp. 1057–1063. MIT Press, 2000.

Syed, Umar and Schapire, Robert E. A reduction from apprenticeship learning to classification. In *NIPS*, 2011.

Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement elarning. *Machine Learning*, 8 (23), 1992.

Xu, Yuehua and Fern, Alan. On learning linear ranking functions for beam search. In *ICML*, pp. 1047–1054, 2007.

Xu, Yuehua, Fern, Alan, and Yoon, Sung Wook. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, pp. 2041–2046, 2007.

Younger, D. H. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, February 1967.

Ziebart, Brian, Maas, Andrew, Bagnell, J. Andrew, and Dey, Anind. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.