Mach Learn
DOI 10.1007/s10994-009-5110-1

Training parsers by inverse reinforcement learning

Gergely Neu · Csaba Szepesvári

Received: 29 April 2008 / Accepted: 20 March 2009
Springer Science+Business Media, LLC 2009

Abstract One major idea in structured prediction is to assume that the predictor computes its output by finding the maximum of a score function. The training of such a predictor can then be cast as the problem of finding weights of the score function so that the output of the predictor on the inputs matches the corresponding structured labels on the training set. A similar problem is studied in inverse reinforcement learning (IRL) where one is given an environment and a set of trajectories and the problem is to find a reward function such that an agent acting optimally with respect to the reward function would follow trajectories that match those in the training set. In this paper we show how IRL algorithms can be applied to structured prediction, in particular to parser training. We present a number of recent incremental IRL algorithms in a unified framework and map them to parser training algorithms. This allows us to recover some existing parser training algorithms, as well as to obtain a new one. The resulting algorithms are compared in terms of their sensitivity to the choice of various parameters and generalization ability on the Penn Treebank WSJ corpus.

Keywords Reinforcement learning · Inverse reinforcement learning · Parsing · PCFG · Discriminative parser training · Parser training · Parsing as behavior

Editor: Charles Parker.

G. Neu (✉)
Department of Computing Science, Budapest University of Technology and Economics,
Műegyetem rakpart 3-9, 1111 Budapest, Hungary
e-mail: neu.gergely@gmail.com

G. Neu · C. Szepesvári
Computer and Automation Research Institute of the Hungarian Academy of Sciences,
Kende utca 13-17, 1111 Budapest, Hungary

C. Szepesvári
Department of Computing Science, University of Alberta, Edmonton T6G 2E8, Alberta, Canada

Published online: 11 April 2009

1 Introduction

In many real world problems the problem is to predict outputs with a non-trivial structure given some inputs (Bakir et al. 2007). A popular approach for training such predictors is to assume that given some input, the structured output is obtained by solving an optimization problem, where the optimization problem depends on the input and a set of tunable weights. The predictor then is learn by tuning the weights so that the predictor's outputs match the targets in a training data available as input-output pairs. Oftentimes dynamic programming is used to solve the optimization problem, e.g., in the classical examples of sequence labeling using HMMs or PCFG parsing (Manning and Schütze 1999), but also in more complex domains such as RNA structure prediction (Rivas and Eddy 1999) or image segmentation (Elliott et al. 1984).

Dynamic programming is also one of the main techniques to find optimal policies in Markov Decision Processes (MDPs). In inverse reinforcement learning (IRL) the goal is to build a model of the behavior of an expert that optimizes the expected long-term cumulated reward in a Markovian environment given a set of trajectories that the expert followed (Ng and Russell 2000). In this framework the expert takes actions that result in (stochastic) state-changes and in each time step the expert incurs some immediate reward which depends on the state of the process and the action taken. The dynamics (of how the states change as a result of executing some action) is assumed to be known, but the immediate rewards are unknown. The IRL task is to find the immediate rewards so that the trajectories that result from following an optimal policy corresponding to the identified rewards matches the observed trajectories.

Returning to structured prediction problems let us assume that the structured outputs are built up in a step-by-step manner. Then the stages of this building process can be viewed as states that the "expert builder" goes through when building the output. Assume that each building step of this process contributes some elementary value to a total score and that the aim of the expert is to maximize this total score. The problem of finding these elementary values can then be viewed as an IRL problem: The unknown values can play the role of immediate rewards, while the rules governing the building process correspond to the dynamics of a controllable process. The main motivation of this paper is to make this connection explicit, allowing one to derive algorithms to train structured predictors from existing IRL methods. For the sake of specificity in this paper we focus on the problem of training parsers that use probabilistic context free grammars (PCFGs).

We will make the connection between IRL and parser training explicit by mapping parsing problems into episodic Markovian Decision Processes (MDPs). In fact, a parse in this framework is obtained as the trajectory when an optimal policy is followed in an appropriately defined MDP. This idea is not completely new: The reverse connection was exploited by Ratliff et al. (2006) who derived an algorithm for inferring rewards using the large margin approach of Taskar et al. (2005). Maes et al. (2007) have used reinforcement learning for solving the structured prediction problem of sequence labeling. The Ph.D. thesis of Daumé III (2006) (and the unpublished work of the same author) presents the more general idea of producing structured outputs by making sequential decisions by decomposing the structured outputs to variable length vectors. Once parsing is represented as a search problem (or sequential decision making problem) one can use any search technique to find a good parse tree (depending on the search problem, dynamic programming might be impractical). This has been recognized long ago in the parsing community: Klein and Manning (2003) proposes an A^* algorithm for retrieving Viterbi parses, Collins and Roark (2004) proposes incremental beam-search, while Turian and Melamed (2006) proposes a uniform-cost search approach. Note that the problem of finding an optimal policy given an MDP is called planning and has a large literature. In this paper, to make the parallel with IRL algorithms clear, we nevertheless restrict ourselves to PCFG parsing when dynamic programming is sufficiently powerful. However, we would like to emphasize once again that the issue of how a good (or optimal) plan (policy) is obtained is independent of the problem of designing an algorithm to find a good reward function for this MDP based on some training data (which corresponds to parser training).

In this paper we consider five IRL algorithms: The first algorithm is the Projection algorithm of Abbeel and Ng (2004), the second is the IRL algorithm of Syed and Schapire (2008), called the Multiplicative Weights for Apprenticeship Learning (MWAL) algorithm and the third is the Policy Matching (PM) algorithm of Neu and Szepesvári (2007). The fourth algorithm considered is the Max-Margin Planning method of Ratliff et al. (2006) which has already been mentioned previously. This algorithm uses the same criterion as Taskar et al. (2004), but instead of using the so-called structured SMO method used by Taskar et al. (2004), following the suggestion of Ratliff et al. (2006) we implement the optimizer using a subgradient method. The last algorithm is the recently proposed IRL method of Ziebart et al. (2008) which turns out to be a close relative to the Maximum Entropy discriminative reranking method proposed by Charniak and Johnson (2005).

One major contribution of the paper is that the IRL algorithms are presented using a unified notation. This allows us to elaborate on similarities and differences between them. In particular, we show how the IRL version of the Perceptron algorithm (Freund and Schapire 1999; Collins 2002; Collins and Roark 2004) can be derived from Max-Margin Planning or that MWAL is the "exponentiated gradient" version of this algorithm. We will also show that the Projection algorithm of Abbeel and Ng (2004) can also be regarded as a special instance of Max-Margin Planning.

The algorithms are compared in extensive experiments on the parser training task using the Penn Treebank WSJ corpus: The experiments were run on a cluster in our institute and took a total of approximately 30,000 hours of CPU time. We test the algorithm's sensitivity to the selection of the step-size sequence and the regularization parameter. We also investigate their generalization ability as a function of the size of the training set. In addition to reporting results on a single hold-out test set (as it is typically done in the parser training literature), we also report results when performance is measured with cross-validation, allowing us to reason about the robustness of the results obtained in the "standard" way. The experiments show that the Max-Margin, MaxEnt and the Policy Matching algorithms (the latter of which is introduced here for parser training for the first time) lead to significantly better results than the other three algorithms (i.e., the Perceptron algorithm, MWAL and the Projection algorithm), while the performance of these latter three algorithms are essentially indistinguishable from each other. We also find that reporting results on a single hold-out test set might lead to conclusions that cannot be supported when testing is done using cross-validation. In particular, when measured on a single hold-out set, the F_1 error reduction of Policy Matching was found to be 26.46%, while the error reduction of the Maximum Entropy method was found to be 20.73%, while when their performances were compared using cross-validation then no statistically significant differences were found.

We have found significant differences between the performance of the training algorithms. This shows that the choice of the parser training method can be crucial for achieving good results. In this paper we decided to deal only with a relatively simple grammatical model. The consequence of this is that in absolute terms our results are not as good as those obtained with more sophisticated grammatical models such those in (Charniak and Johnson

2005; Turian and Melamed 2006; Titov and Henderson 2007). We have made this choice deliberately as this allowed us to compare the methods studied in a thorough manner, hoping the results can serve as a useful guideline for future studies.

The paper is organized as follows: In Sect. 2 we briefly present the required background in parsing (most notably on PCFGs) and the basic concepts of Markov Decision Processes (MDPs). The reduction of PCFG parsing to solving MDPs is presented in Sect. 3. In Sect. 4 we present IRL algorithms in a unified framework. The parser training algorithms derived from the IRL algorithms are presented in Sect. 5. Finally, experimental results are given in Sect. 6 and our conclusions are drawn in Sect. 7.

2 Background

The purpose of this section is to provide a quick introduction to PCFG parsing and MDPs.

2.1 PCFG parsing

In this section we present the formalism used to describe probabilistic context free grammars (PCFGs) in the rest of the paper. The material presented here is based on Manning and Schütze (1999).

A PCFG is a 5-tuple $G = (\mathcal{W}, \mathcal{N}, S, \mathcal{R}, \sigma)$, where

1. $\mathcal{W}$ is set with n_t elements, called the *terminal vocabulary* (i.e., the set of terminal symbols).
2. $\mathcal{N}$ is a set with n_{nt} elements, called the *nonterminal vocabulary* (i.e., the set of nonterminal symbols).
3. The sets $\mathcal{W}$ and $\mathcal{N}$ are disjoint.
4. $S \in \mathcal{N}$ is a *start symbol*.
5. $\mathcal{R}$ is a subset of the set of all possible *production rules*,

$$\mathcal{R}_0 = \{R : R \equiv N \to \xi, N \in \mathcal{N}, \xi \in (\mathcal{N} \cup \mathcal{W})^+\},$$

   where $(\mathcal{N} \cup \mathcal{W})^+$ denotes the set of non-empty words over $\mathcal{N} \cup \mathcal{W}$.
6. $\sigma : \mathcal{R} \to (-\infty, 0]$ is the scoring function that satisfies

$$\sum_{\{R \in \mathcal{R} : \mathrm{lhs}(R) = N\}} \exp(\sigma(R)) = 1, \quad \forall N \in \mathcal{N},$$

   where for $R \in \mathcal{R}$, $R = N \to \xi$, $\mathrm{lhs}(R) = N$, i.e., for any given N, $\exp(\sigma)$ is a probability distribution when it is restricted to production rules with left-hand side N.

From now on we will focus on grammars that are in Chomsky normal form, i.e., all rules are either of the form $R \equiv N \to N_{\mathrm{left}}\ N_{\mathrm{right}}$ ($N_{\mathrm{left}}, N_{\mathrm{right}} \in \mathcal{N}$) or of the form $R \equiv N \to w$ ($w \in \mathcal{W}$).

A PCFG generates a countable set $\mathcal{T}$ of *parse trees* or *parses*. A parse tree $\tau \in \mathcal{T}$ is a set of *constituents*, i.e., triples of form $c = (N_c, start_c, end_c) \in (\mathcal{N} \cup \mathcal{W}) \times \mathbb{N} \times \mathbb{N}$, where $start_c$ and end_c are the respective indices of the first and last words forming the constituent. Formally, $\tau \subset \mathcal{T}_0$, where $\mathcal{T}_0 = \{(N, i, j) : N \in \mathcal{N}, i, j \in \mathbb{N}, i \le j\} \cup \{(w, i, i) : w \in \mathcal{W}, i \in \mathbb{N}\}$ is the set of all possible constituents. In order to qualify as a parse tree a set of constituents has to meet a number of requirements. Before specifying these, we need a few definitions: For integers i, j, let $[i, j] = \{n \in \mathbb{N} : i \le n \le k\}$. Let (N, i, j), (N', i', j') be two constituents. We say that (N, i, j) is an *ancestor* of (N', i', j') (and (N', i', j') is a *descendant* of (N, i, j)) if $[i', j'] \subset [i, j]$. Now, fix a set of consituents τ. We say that $(N, i, j) \in \tau$ is a *parent* of $(N', i', j') \in \tau$ in τ (and (N', i', j') is a *child* of (N, i, j) in τ) if (N, i, j) is an ancestor of (N', i', j') and for every descendant $(N'', i'', j'') \in \tau$ of (N, i, j), either $[i'', j''] \cap [i', j'] = \emptyset$ or $[i'', j''] \subset [i', j']$. Now, a set of constituents, τ, is called a parse-tree if the following hold:

1. There is a single constituent in the parse tree, called the *root*, that has the form $(S, 1, n)$.
2. For all i $(1 \le i \le n)$ a constituent of the form (w, i, i) is in the parse tree, where $w \in \mathcal{W}$.
3. The constituent set τ is linearly ordered, binary and consistent, i.e.,
   (a) Every constituent in τ except the root has a unique parent in τ and is the descendant of the root.
   (b) Every constituent $(N, i, j) \in \tau$ has at most two children in τ.
   (c) If (N, i, j) has a single child c' then it must hold that $i = j$ and $c' = (w, i, i)$ for some $w \in \mathcal{W}$ and $N \to w \in \mathcal{R}$.
   (d) If (N, i, j) has two children, c_{left} and c_{right}, then it must holds that $i < j$, $c_{\text{left}} = (N_{\text{left}}, i, k)$, $c_{\text{right}} = (N_{\text{right}}, k + 1, j)$, and $N \to N_{\text{left}} N_{\text{right}} \in \mathcal{R}$.

A tree is called a *partial parse tree* if it satisfies all the previous properties except item 2. The set of partial parse trees will be denoted by $\mathcal{T}_1$, while the set of parse trees will be denoted by $\mathcal{T}_2$. A constituent $c = (N, start_c, end_c)$ is called *unexpanded* in τ if it has no child in τ. The set of unexpanded constituents of τ is denoted by $U(\tau)$. We say that the rule $R \equiv N \to N_{\text{left}} N_{\text{right}}$ *occurs* in τ if there exists a constituent $c = (N, i, j)$ in τ such that c has two children, c_{left} and c_{right} with non-terminals N_{left} and N_{right}, respectively. The number of *occurrences* of R in τ is defined as

$$f(R, \tau) = \sum_{(N,i,j) \in \tau} \mathbb{I}(\exists i \le k < j : (N_{\text{left}}, i, k), (N_{\text{right}}, k + 1, j) \in \tau). \tag{1}$$

Each parse tree $\tau \in \mathcal{T}$ represents the grammatical structure of a valid sentence in the language generated by the grammar. This sentence is called the *yield* of the tree and will be denoted by y_τ and is defined as $w_1 \cdots w_n$, where n is the unique integer such that $(S, 1, n) \in \tau$ and $w_i \in \mathcal{W}$ is the unique terminal symbol such that $(w_i, i, i) \in \tau$. If $s \in \mathcal{W}^+$ is the yield of τ we also say that τ is a parse-tree of s and s is generated by the grammar G. We will use the notation w_{ab} for the sequence of words $w_a w_{a+1} \cdots w_b$.

A PCFG defines a probability distribution p over all generated parse trees:

$$p(\tau|G) \propto \prod_{R \in \mathcal{R}} \exp(f(R, \tau)\sigma(R)), \tag{2}$$

where $f(R, \tau)$ is the number of occurrences of rule R in parse tree τ.[1] This way we can also define the probability of a sentence $s \in \mathcal{W}^+$ as

$$p(s|G) = \sum_{\{\tau \in \mathcal{T} : y_\tau = s\}} p(\tau|G) = \sum_{\tau \in \mathcal{T}} p(\tau, s|G), \tag{3}$$

[1] If there are non-terminals in G whose expansion cannot produce valid sentences of the terminals then it can happen that $\sum_{\tau \in \mathcal{T}} p(\tau|G) < 1$. However, in this case G is in some way incomplete. In this paper we shall not deal with grammars with such "dangling" non-terminals, so we can safely disregard this issue.

where $p(\tau, s|G) = \mathbb{I}(y_\tau = s)p(\tau|G)$. The probability that τ is a parse-tree for sentence s is given by:

$$p(\tau|s, G) = \frac{p(\tau, s|G)}{p(s|G)}.$$

The problem of parsing is to find the most probable parse for a sentence s, given a grammar G. This can be formalized as finding the parse

$$\tau^* = \arg\max_{\tau\in\mathcal{T}} p(\tau|s, G) = \arg\max_{\tau\in\mathcal{T}} \frac{p(\tau, s|G)}{p(s|G)} = \arg\max_{\tau\in\mathcal{T}} p(\tau, s|G),$$

which can be further written as

$$\begin{aligned}\tau^* &= \arg\max_{\tau\in\mathcal{T}} \left\{ \mathbb{I}(y_\tau = s) \prod_{R\in\mathcal{R}} \exp(f(R,\tau)\sigma(R)) \right\} \\ &= \arg\max_{\tau\in\mathcal{T}} \left\{ \log \mathbb{I}(y_\tau = s) + \sum_{R\in\mathcal{R}} f(R,\tau)\sigma(R) \right\}. \qquad (4)\end{aligned}$$

Then, parsing can be viewed as finding the parse tree with maximal score among the trees that yield s. The tree τ^* is called the *maximum scoring tree*.

2.2 Markovian decision processes

Markov decision processes (MDPs) are a widely studied, general mathematical framework for sequential decision problems (e.g., Bertsekas and Tsitsiklis 1996). The essential idea is that an agent interacts with its environment, changing the state of the environment and receiving a sequence of rewards. The goal of the agent is to maximize the cumulative sum of the rewards received. Here, we shall only deal with episodic MDPs, i.e., MDPs that have terminal states. Formally, a countable episodic MDP is defined with a 5-tuple $M = (\mathcal{X}, \mathcal{A}, T, \mathcal{X}_T, r)$, where

$\mathcal{X}$ is a countable set of *states*.
$\mathcal{A}$ is a finite set of *actions*.
T is the *transition function*; $T(x'|x,a)$ stands for the probability of transitioning from state x to x' upon taking action a ($x, x' \in \mathcal{X}, a \in \mathcal{A}$).
$\mathcal{X}_T$ is the set of terminal states: upon reaching a state in this set the process terminates.
r is the *reward function*; $r : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$. This function determines the reward upon selecting action $a \in \mathcal{A}$ at state $x \in \mathcal{X}$.

An MDP is called deterministic if all the transitions are deterministic, i.e., if for any $(x, a) \in \mathcal{X} \times \mathcal{A}$, $T(x'|x, a) = 0$ holds for all next states $x' \in \mathcal{X}$ except one state. In such cases, by slightly abusing the notation, we write $T(x, a)$ to denote the one next state.

A *stochastic stationary policy* (in short: policy) is a mapping $\pi : \mathcal{A} \times \mathcal{X} \to [0, 1]$ satisfying $\sum_{a\in\mathcal{A}} \pi(a|x) = 1, \forall x \in \mathcal{X}$.[2] The value of $\pi(a|x)$ is the probability of taking action a in state x. For a fixed policy, the *value* of a state $x \in \mathcal{X}$ is defined by

$$V_r^\pi(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{H-1} r(x_t, a_t) \middle| \, x_0 = x \right].$$

[2] Instead of $\pi(a, x)$ we use $\pi(a|x)$ to emphasize that $\pi(\cdot, x)$ is a probability distribution.

Here x_t is a random process that is obtained by following policy π in the MDP. This means that in each time step, x_{t+1} follows the distribution $T(\cdot|x_t, a_t)$, where a_t is a random action drawn from $\pi(\cdot|x_t)$. In particular, given x_t and a_t, x_{t+1} is independent of the states, actions and rewards that were obtained prior time step t. This is called the *Markov property*. Further, it is assumed that x_0 is such that $D(x) > 0$ for some initial distribution $D(x) = P(x_0 = x)$. The number H in the above equation is the first (random) time when the process enters the set of terminal states ($x_H \in \mathcal{X}_T$ and for $t < H$, $x_t \notin \mathcal{X}_T$). (The notation $\mathbb{E}_\pi$ is used to signify that the expectations is taken by assuming that the transitions are generated while following π.) The function $V_r^\pi : \mathcal{X} \to \mathbb{R}$ is well-defined and is called the *value function* corresponding to policy π and reward function r.

A policy that maximizes the values at all states is called an *optimal policy* and is denoted by π_r^*. Thus, an optimal policy maximizes the cumulated total expected reward irrespective of where the process starts. The values under an optimal policy define the *optimal value function*. In fact, the optimal value function also satisfies

$$V_r^*(x) = \max_\pi V_r^\pi(x), \quad x \in \mathcal{X}.$$

The *Q-functions* (or Q-factors, *action-value functions*) can be defined similarly:

$$Q_r^\pi(x, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{H-1} r(x_t, a_t) \middle| x_0 = x, a_0 = a \right].$$

In words, the value of a at x under π is the expected total reward assuming that the first action taken in $x \in \mathcal{X}$ is $a \in \mathcal{A}$, and the further actions are obtained by following π. Similarly to the optimal state-values, the *optimal action-values* or *optimal Q-factors* are given by maximizing the action values with respect to the policies: $Q_r^* = Q_r^{\pi_r^*}$. Further,

$$Q_r^*(x, a) = \max_\pi Q_r^\pi(x, a), \quad (x, a) \in \mathcal{X} \times \mathcal{A}.$$

It is also useful to define the *advantage functions*:

$$A_r^\pi(x, a) = Q_r^\pi(x, a) - V_r^\pi(x), \quad (x, a) \in \mathcal{X} \times \mathcal{A}.$$

Similarly, $A_r^* = Q_r^* - V_r^*$. The significance of advantage (and action-value) functions is that knowing the optimal advantage (action-value) function allows one to behave optimally: π is an optimal policy if it holds that for any state x, π picks an action that maximizes $A_r^*(x, a)$. Accordingly, such actions will be called optimal.

One variation of MDPs that we will need is when not all actions are available in each state. In that case we use $\mathcal{A}(x)$ to signify the set of actions that are *admissible* in state x. The rest of the definitions need then to be adjusted accordingly (i.e., policies in state x cannot choose actions outside of $\mathcal{A}(x)$).

Given a class of MDPs $M_i = (\mathcal{X}_i, \{\mathcal{A}_i(x)\}, T_i, \mathcal{X}_{T,i}, r_i)$ $(i = 1, 2, \ldots)$ with disjoint state spaces the *union* of these MPD is defined to be $M = (\mathcal{X}, \{\mathcal{A}(x)\}, T, \mathcal{X}_t, r)$ as follows: $\mathcal{X} = \bigcup_i \mathcal{X}_i$ and $\mathcal{X}_T = \bigcup_i \mathcal{X}_{T,i}$. Now pick $x \in \mathcal{X}$. Then there exists a unique index i such that $x \in \mathcal{X}_i$. Pick this index i. Then $\mathcal{A}(x) = \mathcal{A}_i(x)$ and for $a \in \mathcal{A}(x)$, $T(\cdot|x, a) = T_i(\cdot|x, a)$ and $r(x, a) = r_i(x, a)$.

We will also need the definition of legal trajectories: An alternating sequence of states and actions, $\xi = (x_0, a_0, x_1, a_1, \ldots, x_{H-1})$, is a legal trajectory if for any $0 \le i \le H - 2$, a_i is admissible in x_i, $T(x_{i+1}|x_i, a_i)$ is positive and x_{H-1} is a terminal state. In other words, a legal trajectory is a sequence of states and actions that can be obtained by following some policy in the MDP. The set of legal trajectories shall be denoted by Ξ.

3 PCFG parsing as an MDP

The purpose of this section is to show that finding the best parse in a PCFG is equivalent to following an optimal policy in an appropriately defined deterministic, episodic MDP.

To make the connection to MDPs clear, notice that following an optimal policy in a deterministic, episodic MDP from a given initial state is equivalent to finding a path that connects the initial state to a terminal state such that the total reward along the path is maximal. Now, in PCFG parsing the aim is to construct a parse with maximal total score, where the scores of the individual rules are additively combined (cf. (4)). The idea is that this parse tree can be constructed in a sequential manner, i.e., starting from the sentence symbol and then expanding the obtained partial parse trees by applying appropriate production rules until the sentence is obtained in the leaf nodes of the tree. This process corresponds to a *top-down* construction of the parse tree. (Other construction orders are also possible, but are not considered here.) Below we illustrate the process with an example (cf. Fig. 1).

Definition 1 The *top-down parsing MDP* for sentence w_{1K} and grammar $G = \{\mathcal{N}, \mathcal{W}, S, \mathcal{R}, \sigma\}$ is a 5-tuple $M^G_{w_{1K}} = (\mathcal{X}, \{\mathcal{A}(x)\}, T, \mathcal{X}_T, r)$ defined as follows:

$\mathcal{X}$ is the state space consisting of states that represent partial parse trees with root $(S, 1, K)$: $\mathcal{X} = \{\tau \in \mathcal{T}_1 : (S, 1, K) \in \tau\}$.

$\mathcal{A}(x)$ is the set of admissible actions in x, with elements that are in the form of triplets with the following components: an unexpanded constituents of x, a rule of the grammar and a splitting point. Formally,

$$\begin{aligned}
\mathcal{A}(x) &= \mathcal{A}_1(x) \cup \mathcal{A}_2(x),\\
\mathcal{A}_1(x) &= \{(c, R, \mathit{split}) : c = (N_c, \mathit{start}_c, \mathit{end}_c) \in U(x),\\
&\qquad R \equiv N_c \to N_{\text{left}} N_{\text{right}} \in \mathcal{R}, \mathit{start}_c \le \mathit{split} < \mathit{end}_c\},\\
\mathcal{A}_2(x) &= \{(c, R, \mathit{split}) : c = (N_c, \mathit{start}_c, \mathit{start}_c) \in U(x),\\
&\qquad R \equiv N_c \to w \in \mathcal{R}, \mathit{split} = \mathit{start}_c\}.
\end{aligned}$$

The components of a specific action a will be referred to as c^a, R^a and split^a (where this is appropriate).

T the transition function is deterministic: Fix $x \in \mathcal{X}$, $a = (c, R, \mathit{split}) \in \mathcal{A}_1(x)$, where $R \equiv N_c \to N_{\text{left}} N_{\text{right}}$. Then the next state $x' = T(x, a)$ is obtained by adding the new constituents $c_{\text{left}} = (N_{\text{left}}, \mathit{start}_c, \mathit{split})$ and $c_{\text{right}} = (N_{\text{right}}, \mathit{split} + 1, \mathit{end}_c)$ to x: $x' = x \cup \{c_{\text{left}}, c_{\text{right}}\}$. If $a = (c, R, \mathit{split}) \in \mathcal{A}_2(x)$, where $R \equiv N \to w$, then the new state x' is obtained by adding the constituent $c' = (w, \mathit{start}_c, \mathit{start}_c)$ to x: $x' = x \cup \{c'\}$.

$\mathcal{X}_T$ the set of terminal states are those states x where the set of admissible actions, $A(x)$, is empty.

r the reward function is defined as follows: It is a function $r : \mathrm{Dom}_r \to \mathbb{R}$, where $\mathrm{Dom}_r = \{(x, a) : x \in \mathcal{X}, a \in A(x)\}$. The reward of an action that leads to a transition to a terminal state x_T is $-\infty$ if x_T does not correspond to a full parse tree of w_{1K}. For all other states, $r(x, a) = \sigma(R^a)$.

In what follows when the grammar is clear from the context, we will drop the superscript G. It is easy to see that any policy in these MDPs terminates after a finite number of steps. When needed we annotate the states of $M_{w_{1K}}$ by w_{1K}. If we do so, the MDPs corresponding to different sentences will have disjoint state spaces. Thus, we can take the union of these MDPs, which in turns defines the MDP corresponding to G:

Definition 2 Let $G = \{\mathcal{N}, \mathcal{W}, S, \mathcal{R}, \sigma\}$ be a grammar. Then the top-down parsing MDP corresponding to G, $M_G = (\mathcal{X}, \{A(x)\}, T, \mathcal{X}_T, r)$ is obtained as the disjoint union of the MDPs corresponding to all sentences generated by G.

Although an initial state distribution is not part of the definition of MDPs, sometimes we will need such a distribution. Note that a PCFG naturally gives rise to an initial state distribution: In fact, the natural distribution D assigns non-zero probabilities only to states of the form $x_{(0,w_{1K})} = \{(S, 1, K), w_{1K}\}$, where $w_{1K} \in \mathcal{W}^+$ is a sentence generated by the grammar G. In particular, $D(x_{(0,w_{1K})}) = p(w_{1K}|G)$, where $p(w_{1K}|G)$ is determined as in (3). For the sake of simplicity, when it is clear from the context which MDP we are in, we will use the unannotated symbols.

The idea of state construction is illustrated in Fig. 1 which shows the state

$$x_0 = \{\text{“It was love at first sight”}, (\mathrm{S}, 1, 6), (\mathrm{NP}, 1, 1), (\mathrm{PRP}, 1, 1), (\mathrm{VP}, 2, 6)\}$$

in the form of a tree, three actions that are admissible in this state, namely,

$$a_1 = ((\mathrm{VP}, 2, 6), \mathrm{VP} \to \mathrm{VBD\ NP}, 2),$$
$$a_2 = ((\mathrm{PRP}, 2, 2), \mathrm{PRP} \to \text{“it”}, 2),$$
$$a_k = ((\mathrm{VP}, 2, 6), \mathrm{VP} \to \mathrm{VP\ PP}, 3),$$

and the three states resulting from applying the respective actions.

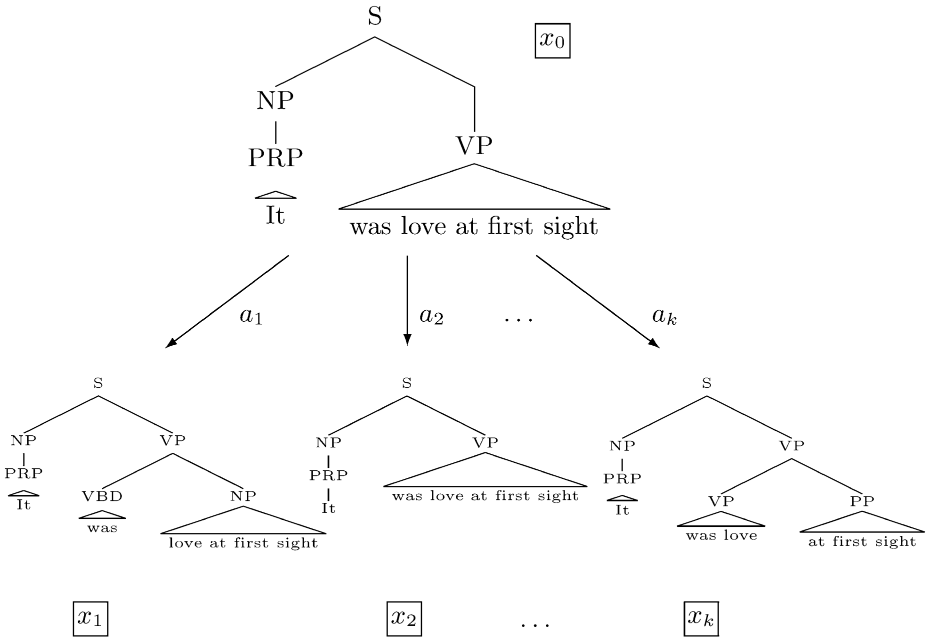


Fig. 1 Illustration of states and transitions in an MDP assigned to a CFG. The states $(x_0, x_1, x_2, \ldots, x_k)$ are partial parse trees and the actions $(a_1, a_2, \ldots, a_k)$ correspond to valid extensions of the parse trees where the action is applied. For more explanation see the text

The rewards associated with taking these actions in this state are:

$$r(x_0, a_1) = \sigma(\text{VP} \to \text{VBD NP}),$$

$$r(x_0, a_2) = \sigma(\text{PRP} \to \text{“it”}),$$

$$r(x_0, a_k) = \sigma(\text{VP} \to \text{VP PP}).$$

The following result follows immediately from the above construction and hence its proof is omitted:

Proposition 1 *Let $G = \{\mathcal{N}, \mathcal{W}, S, \mathcal{R}, \sigma\}$ be a grammar and M_G be the corresponding top-down parsing MDP. Let π^* be an optimal policy in M_G. Pick some sentence $w_{1K} \in \mathcal{W}^+$ and let $x^*_{w_{1K}} = \{w_{1K}\} \cup \tau$ be the terminal state reached by π^* when started in M_G from state $x_0 = \{w_{1K}, (S, 1, K)\}$. Then the following hold: if w_{1K} is generated by the grammar G then $x^*_{w_{1K}}$ will not be a failure state and $V^*_r(x_0) = \log p(\tau, w_{1K}|G)$, otherwise $x^*_{w_{1K}}$ will be a failure state and $V^*_r(x_0) = -\infty$.*

An optimal policy can be recovered from the knowledge of the optimal advantages. The next statement shows that the optimal advantages at a state can be computed if one calculates the optimal state values in a problem with a larger state space. This new MDP for a given sentence w_{1K} generated by G is obtained as follows: $\hat{M}_{w_{1K}} = \bigcup_{N \in \mathcal{N}, 1 \le i \le j \le K} M_{w_{1K},(N,i,j)}$, where in $M_{w_{1K},(N,i,j)}$ the set of admissible actions, the transitions, the terminal states and rewards are defined as it was done for $M_{w_{1K}}$, just the state space of $M_{w_{1K},(N,i,j)}$ is obtained by recursively following the transitions resulting from admissible actions and when the initial state is taken to be $\{(N, i, j)\}$.[3] The special states, $\{(N, i, j)\}$, in the state space of $\hat{M}_{w_{1K}}$ will be called the initial states in $\hat{M}_{w_{1K}}$. The following proposition holds:

Proposition 2 *Let $w_{1K} \in \mathcal{W}^*$ and consider $\hat{M}_{w_{1K}}$. Pick x in the state space of $\hat{M}_{w_{1K}}$ and $a \in \mathcal{A}(x)$. Assume that $a = (c, R, split)$, where $R \equiv N \to N_{\text{left}} N_{\text{right}}$ and let*

$$x_N = \{(N, start_c, end_c)\},$$

$$x_{\text{left}} = \{(N_{\text{left}}, start_c, split)\},$$

$$x_{\text{right}} = \{(N_{\text{right}}, split + 1, end_c)\}$$

be states of $M'_{w_{1K}}$. Let $\hat{V}^*_r$ be the optimal value function in $\hat{M}_{w_{1K}}$ and let $\hat{A}^*_r$ be the corresponding advantage function. Then

$$\hat{A}^*_r(x, a) = \hat{r}(x, a) + \hat{V}^*_r(x_{\text{left}}) + \hat{V}^*_r(x_{\text{right}}) - \hat{V}^*_r(x_N), \tag{5}$$

where $\hat{r}$ is the reward function in $\hat{M}_{w_{1K}}$.

Proof We have $\hat{A}^*_r(x, a) = \hat{r}(x, a) + \hat{V}^*_r(x') - \hat{V}^*_r(x)$, where $x' = T(x, a)$ is the state obtained when a is applied in x. Then we have $\hat{V}^*_r(x') - \hat{V}^*_r(x) = \hat{V}^*_r(x_{\text{left}}) + \hat{V}^*_r(x_{\text{right}}) -$

[3] Thus a state in the extended MDP $\hat{M}_{w_{1K}}$ corresponds to a partial or full subtree spanning a number of words in w_{1K}, while a state in $M_{w_{1K}}$ corresponds to a partial or full subtree spanning the *full sentence* w_{1K}.

$\hat{V}_r^*(x_N)$ thanks to the additive rewards, that the effects of the actions are local and the construction of $x_{\text{left}}, x_{\text{right}}$ and x_N. Combining the first and last equations in the proof and reordering the terms gives the result. □

As a corollary of this proposition we get that the optimal advantage function of $M_{w_{1K}}$ can be obtained by computing the optimal values of the initial states in $\hat{M}_{w_{1K}}$. To see this assume that $x \in M = M_{w_{1K}}$ and consider $a \in \mathcal{A}(x)$. Let $x' = T(x, a)$. Assume that $a = ((N, i, j), R, \mathit{split})$ and let $\tilde{M} = M_{w_{ij},(N,i,j)}$. If $i = j$, $R \equiv N \to w$ with some $w \in \mathcal{W}$ then the statement holds trivially. Hence, consider the case when $i < j$, $R \equiv N \to N_{\text{left}} N_{\text{right}}$. Let $\tilde{A}_r^*(x, a)$ denote the optimal advantage function in $\tilde{M}$ and let $\tilde{V}$ be the optimal state value function in $\tilde{M}$. Then if y denotes $\{(N, i, j)\} \in \tilde{M}$ then $a \in \mathcal{A}(y)$ (a is admissible in $\tilde{M}$ at y). Clearly, the advantage of a at x is the same as the advantage of a at y (in $\tilde{M}$): $A_r^*(x, a) = \tilde{A}_r^*(y, a)$. Now, by the above proposition, $\tilde{A}_r^*(y, a)$ can be obtained by from the optimal values of the initial states in $\hat{M}_{w_{1K}}$, proving the claim.

In parsing the optimal values assigned to initial states of $\hat{M}_{w_{1K}}$ are called "inside Viterbi-scores". In fact, Viterbi-parsers (a.k.a., CKY parsers) compute the optimal parse tree by first computing these scores (clearly, knowing the optimal advantage function is sufficient to recover optimal parses). It follows that the optimal advantage function in the MDP $M_{w_{1K}}$ can be computed in $\mathcal{O}(K^3 n_{nt}^3)$ time in the worst case, see e.g. Collins (1999).

4 A unified view of inverse reinforcement learning algorithms

In this subsection we present an overview of current Inverse Reinforcement Learning (IRL) methods. First, we give the definition of the IRL problem and discuss some of its difficulties. We then present five existing algorithms in a unified notation. The unified notation allows us to compare these algorithms and elaborate on their similarities and differences.

Informally, IRL is a method to build a model for the observed behavior of an expert by finding the definition of the task that the expert performs. Assuming that the expert acts in an MDP, this can be stated more formally as finding the reward function that generates an optimal behavior that is close enough to the behavior of the expert.[4] This definition still leaves one question open: how do we decide if two particular behaviors are "close enough"? The main difference between the algorithms to be shown is this definition of closeness: once this definition is fixed, we are left with the task of finding an algorithm to efficiently minimize it.

IRL is a difficult problem. Besides the dilemma of selecting an appropriate feature set we have to be aware of that the IRL problem is ill-posed: infinitely many reward functions can give rise to a specific behavior. Even worse, this set of solutions contains degenerate solutions as well, such as the reward function that is identically zero in all state-action pairs (this reward makes all policies optimal). One solution is to give preference to reward functions that robustly generate the observed behavior. An even stronger requirement is that the observed behavior be the *only* optimal behavior with respect to the reward function. The dissimilarity functions should be chosen to encourage such solutions.

[4] The earliest precursor of IRL was the *inverse optimal control* problem, where a linear, time-invariant system is considered with a quadratic cost function. For details and further references, see Boyd et al. (1994).

4.1 A Unified view

Here we present a unified framework for the design of IRL algorithms. An IRL algorithm receives a rewardless MDP $M \setminus r$ (an MDP without a reward function) and a list of trajectories, $\mathcal{D} = \{\xi_1, \ldots, \xi_{N_{\text{traj}}}\}$, that are obtained while an expert follows its policy in the MDP. The task is to come up with a reward function such that trajectories that one obtains by following the optimal policy in the obtained MDP become "close" to the observed trajectories. In order to specify what we mean by "close" we define a dissimilarity function $J = J(r; \mathcal{D})$ that maps reward functions and datasets into reals, assigning higher values to pairs when the optimal behavior with respect to the selected reward function r is less similar to the expert's observed behavior as represented by the dataset $\mathcal{D}$.

Given J, a good reward function r should minimize the dissimilarity between r and $\mathcal{D}$. Thus, one might be interested in computing

$$r^* = \arg\min_r J(r; \mathcal{D}) = ?$$

Below we will argue that all IRL algorithms aim to solve an optimization of this form. In particular, in all these approaches the reward function is sought in a linear form:

$$r_\theta(x,a) = \sum_{i=1}^{n} \theta_i \phi_i(x,a) = \theta^T \phi(x,a), \quad (x,a) \in \mathcal{X} \times \mathcal{A}, \tag{6}$$

where $\phi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$ is a feature extractor. Unless otherwise stated, in this paper we shall consider linear parameterizations only. Given a parameterization the problem becomes to find a parameter vector θ^* such that $J(\theta; \mathcal{D}) = J(r_\theta; \mathcal{D})$ is minimized at θ^*.

The optimal parameter vector is typically found by incremental algorithms. We will see that these algorithms take the form

$$\theta_{k+1} = g(g^{-1}(\theta_k) + \alpha_k \Delta_k),$$

where α_k is the step size at iteration k, g is the so-called link-function (see Warmuth and Jagota 1997) and Δ_k is the parameter update used in the considered IRL method at iteration k. In particular, $g(x) = \exp(x)$ leads to multiplicative, $g(x) = x$ leads to additive updates. The discussion of the relative advantages of these choices is out of the scope of this paper and the interested reader is referred to Cesa-Bianchi and Lugosi (2006), where algorithms of this form are extensively discussed in the online learning framework.

Before moving on to discussing specific IRL algorithms, we need to fix some more notations: The *conditional feature expectation function* with respect to a reward function r is defined by

$$\Phi_r(x,a) = \mathbb{E}_{\pi_r}\left[\sum_{t=0}^{H} \phi(x_t,a_t) \middle| x_0 = x, a_0 = a \right], \quad (x,a) \in \mathcal{X} \times \mathcal{A}, \tag{7}$$

where π_r is an optimal policy w.r.t. r and $(x_0, a_0, x_1, a_1, \ldots)$ is a random trajectory obtained such that $(x_0, a_0) \sim D$ for some distribution D such that $D(x,a) > 0$ holds for any $(x,a) \in \mathcal{X} \times \mathcal{A}$, and $x_{t+1} \sim P(\cdot|x_t, a_t)$, $a_t \sim \pi_r(\cdot|x_t)$, $t \geq 1$.[5] The (unconditional) feature

[5] If there are multiple optimal policies, we pick one in some specific manner (i.e., randomize uniformly across the optimal actions) to make Φ_r well-defined.

expectations are defined by taking the expectation of the conditional feature expectations:

$$\bar{\Phi}_r = \mathbb{E}[\Phi_r(x,a)|x \sim D_0, a \sim \pi_r(\cdot|x)], \tag{8}$$

where D_0 is some initial state distribution.

For the sake of brevity, but at the price of slightly abusing the notation the optimal policy w.r.t. $r = r_\theta$ will be denoted by π_θ and the corresponding state visitation frequencies will be denoted by μ_θ. Similarly, the feature expectations generated by following π_θ will be denoted by Φ_θ:

$$\Phi_\theta \overset{\text{def}}{=} \Phi_{r_\theta}.$$

4.2 Some IRL algorithms

In this section we will examine five different algorithms in the above framework. We shall not deal with the derivation of these algorithms or their convergence properties.

4.2.1 Projection

The Projection algorithm was proposed in Abbeel and Ng (2004). This is the earliest IRL algorithm discussed in this paper.[6]

Dissimilarity Assume that the length of the jth trajectory of $\mathcal{D}$ is H_j and in particular $\xi_i = (x_{t_i+j}, a_{t_i+j})_{j=1}^{H_i}$, where $t_i = \sum_{k=0}^{i-1} H_k$. The estimate of the expert's feature expectation vector is then

$$\bar{\Phi}_E := \frac{1}{N_{\text{traj}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{j=1}^{H_i} \phi(x_{t_i+j}, a_{t_i+j}).$$

Using this notation, the dissimilarity is

$$J(\theta; \mathcal{D}) = \|\bar{\Phi}_\theta - \bar{\Phi}_E\|^2, \tag{9}$$

i.e., the goal of the algorithm is to match the feature expectations underlying the optimal policy and the observed feature expectations.

As noted by Abbeel and Ng (2004), the problem with this dissimilarity is that it can be sensitive to the scaling of the features (see also the discussion by Neu and Szepesvári 2007). Since the algorithm can lead to wildly differing reward functions (and policies) depending on the scaling of the features, this algorithm should be used carefully when the scaling of the features is unknown initially. This remark applies to all the other algorithms presented here, except Policy Matching and the MaxEnt methods which avoid this issue by measuring distances between distributions. In parsing binary features are a natural choice, hence the scaling issue is less of a problem.

[6] The term "inverse reinforcement learning" was first used by Ng and Russell (2000), but that paper did not present a practical IRL algorithm, its aim is mainly to characterize the solution set for the IRL problem.

Update step The parameter updates are done additively at each step ($g(x)=x$), the update vector at the k-th step is

$$\Delta_k = \beta_k(\bar{\Phi}_E - \bar{\Phi}_{\theta_k}) - \beta_k\theta_k \tag{10}$$

where β_k is a special step-size parameter (and the global step-size parameter α_k is kept constant at a value of 1). To compute this step size, we need to maintain a vector Ψ_k throughout the training steps. By setting $\Psi_0 = \bar{\Phi}_{\theta_0}$, the values of β_k and Ψ_k ($k \geq 0$) are computed incrementally in the following way:

$$\beta_k = \frac{(\bar{\Phi}_{\theta_k} - \Psi_{k-1})^T(\bar{\Phi}_E - \Psi_{k-1})}{(\bar{\Phi}_{\theta_k} - \Psi_{k-1})^T(\bar{\Phi}_{\theta_k} - \Psi_{k-1})}, \tag{11}$$

$$\Psi_k = \Psi_{k-1} + \beta_k(\bar{\Phi}_{\theta_k} - \Psi_{k-1}). \tag{12}$$

The original algorithm includes a post-processing step, when a mixed policy is constructed that produces feature expectations with minimal distance to the expert's observed feature expectations. As we want to use this algorithm for parser training, we will not apply this step, as we are interested in only deterministic parsers (i.e., parsers that return the same parse tree for a specific sentence every time it is queried). Instead, as with the other algorithms, we will monitor the performance on a validation set and choose the parameter that gives the best results there.

4.2.2 MWAL

The *multiplicative weights algorithm for apprenticeship learning* (MWAL) was proposed by Syed and Schapire (2008) with the aim to improve the Projection algorithm of Abbeel and Ng (2004).

Dissimilarity In this case the dissimilarity is

$$J(\theta;\mathcal{D}) = \theta^T(\bar{\Phi}_{r_\theta} - \bar{\Phi}_E), \tag{13}$$

where θ is restricted to nonnegative values, corresponding to the assumption that the features are positively correlated with the rewards.[7]

The rationale underlying this criterion is that $\theta^T\bar{\Phi}_{r_\theta}$ can be shown to be the average expected reward under the optimal policy corresponding to r_θ and if the initial states are selected from the distribution D. Further, $\theta^T\bar{\Phi}_E$ can be viewed as an approximation to the average reward that would have been collected by the expert if the reward function was r_θ. The minimization problem corresponds to a robust (minimax) approach: The optimal choice of θ makes the performance of the optimal policy the least favorable compared with that of the expert. Syed and Schapire (2008) show that by von Neumann's minmax theorem the value of J at the minimum is positive. It follows that the found behavior will be better than the expert's behavior even when the least favorable parameters are taken. This makes the algorithm more robust.

[7] Abbeel and Ng (2004) also propose a Max-Margin algorithm that attempts to minimize the same criterion.

Update step The updates proposed to solve this optimization are multiplicative, i.e., $g(x) = \exp(x)$. Further,

$$\Delta_k = \bar{\Phi}_E - \bar{\Phi}_{r_{\theta_k}}. \tag{14}$$

As it is well-known, multiplicative weights algorithms can be sensitive to the choice of step sizes, hence in this work we will compare several choices.[8] The algorithm as proposed originally has performance guarantees for a *randomized* policy which is obtained by randomizing over the policies obtained in the iterations. Again, instead of randomizing, we will use the optimal policy (parser) corresponding to the final parameter vector found by the algorithm when testing its performance.

4.2.3 Max-Margin Planning

This algorithm was published in Ratliff et al. (2006) and is derived from the Max Margin Planning algorithm of Taskar et al. (2005). Note that Ratliff et al. (2006) argue that their problem is distinct from IRL since it concerns a series of planning problems. However, by assuming that the state spaces are disjoint we can take the union of the resulting MDPs (as was done in setting up the top-down parsing MDP corresponding to a grammar). This way solving a sequence of planning problems becomes equivalent to solving a single MDP.

Dissimilarity Let the state-action visitation frequencies, μ_E, and a loss function, $\ell : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^+$, be defined as follows:

$$\begin{aligned} \mu_E(x,a) &:= \frac{\sum_{t=1}^{N} \mathbb{I}(x_t = x \wedge a_t = a)}{N}, \qquad && (x,a) \in \mathcal{X} \times \mathcal{A}, \\ \ell(x,a) &:= c_\ell \mu_E(x,a), && (x,a) \in \mathcal{X} \times \mathcal{A}. \end{aligned}$$

In the above formula, c_ℓ is a positive *loss constant*, which is a parameter of the algorithm. The chosen dissimilarity is the following:

$$J(\theta; \mathcal{D}) = \left(\sum_{x,a} (r_\theta(x,a) - \ell(x,a)) \mu_{\theta,\ell}(x,a) - \sum_{x,a} r_\theta(x,a) \mu_E(x,a) \right) + \frac{\lambda}{2} \|\theta\|_2^2.$$

Here $\mu_{\theta,\ell}$ is the stationary distribution (visitation frequencies/counts in episodic MDPs) generated by the policy that is optimal w.r.t. $r_\theta - \ell$, and $\lambda \geq 0$ is a regularization constant whose role is to control the complexity of the solutions. The role of the loss function is to enforce that the solution found is better (cf. (13)) than other solutions by at least a margin proportional to this loss. Accordingly, here the average expected payoff of the optimal policy corresponding to $r_\theta - \ell$ (and not to r_θ) is compared with the average payoff of the expert. By choosing the loss proportional to the state-visitation frequencies we force rewards of highly visited state-action pairs to take on larger values, encouraging the learn policy to visit such states more oftern. This also has an effect of enforcing meaningful solutions to the IRL problem. In particular, the degenerate solution $\theta = 0$ does not minimize the criterion.

[8] Note that Syed and Schapire (2008) propose a specific step-size sequence for which they can derive theoretical guarantees. However, as shown by our preliminary experiments, in practice this step-size sequence does not perform very well and hence we will not include it in our comparison.

Update step The update of the subgradient algorithm of Ratliff et al. (2006) uses $g(x) = x$ and

$$\Delta_k = \sum_{x,a} \phi(x,a)[\mu_E(x,a) - \mu_{\theta_k,\ell}(x,a)] - \lambda\theta_k = \bar{\Phi}_E - \bar{\Phi}_{r_{\theta_k}-\ell} - \lambda\theta_k. \tag{15}$$

4.2.4 Policy matching

This algorithm directly aims to minimize the distance to the expert's policy (Neu and Szepesvári 2007).

Dissimilarity Assume that $\mathcal{D} = \{(x_1,a_1),\ldots,(x_N,a_N)\}$. Let us build an estimate $\hat{\pi}_E$ of the expert's policy:

$$\hat{\pi}_E(a|x) = \frac{\sum_{t=1}^N \mathbb{I}(x_t = x \wedge a_t = a)}{\sum_{t=1}^N \mathbb{I}(x_t = x)}, \quad (x,a) \in \mathcal{X} \times \mathcal{A}$$

(if a state is not visited by the expert, $\hat{\pi}_E(\cdot|x)$ could be defined arbitrarily). We will also need the empirical state visitation frequencies of the expert:

$$\hat{\mu}_E(x) = \frac{1}{N} \sum_{t=1}^N \mathbb{I}(x_t = x), \quad x \in \mathcal{X}.$$

Then the dissimilarity is given by the formula

$$J(\theta;\mathcal{D}) = \frac{1}{2} \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} \hat{\mu}_E(x)[\pi_\theta(a|x) - \hat{\pi}_E(a|x)]^2. \tag{16}$$

Clearly, this objective function is very different from the previous ones: The aim here is to directly match the behavior and the rewards are used only for parameterizing the class of policies available. Thus one expects this objective to work better when there is not much noise in the observed behavior. One problem with this objective function is that the optimization is convex only in the special case when the expert behavior is deterministic and $\mu_E(x) \neq 0$ holds for all state x. In such a case one can write up an equivalent quadratic program with linear constraints.

Update step We apply a gradient algorithm for minimizing the distance (16). We chose π_θ to be the so-called *Boltzmann-policy* with respect to Q^*_θ:

$$\pi_\theta(\cdot|x) = B(Q^*_\theta(x,\cdot),\eta), \quad B(Q^*_\theta(x,\cdot),\eta)(a) = \frac{\exp(\frac{Q^*_\theta(x,a)}{\eta})}{\sum_{b\in\mathcal{A}(x)} \exp(\frac{Q^*_\theta(x,b)}{\eta})}, \tag{17}$$

where $\eta > 0$ is a "temperature" parameter. The smaller η is, the closer π_θ is to an optimal policy. The reason of not relying on the optimal policy is to make the policy a differentiable function of $Q^*_\theta(x,\cdot)$. The update is additive ($g(x) = x$) and uses

$$\Delta_k = \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} \hat{\mu}_E(x)(\hat{\pi}_E(a|x) - \pi_{\theta_k}(a|x))\partial_\theta \pi_{\theta_k}(a|x), \tag{18}$$

where $\partial_\theta \pi_\theta(a|x)$ is the gradient of $\pi_\theta(a|x)$. As shown by Proposition 4 in Neu and Szepesvári (2007), for almost all θ, the gradient of π_θ can be computed as

$$\partial_\theta \pi_\theta(a|x) = \frac{\pi_\theta(a|x)}{\eta}\left(\Phi_\theta(x,a) - \sum_{b\in\mathcal{A}(x)} \pi_\theta(b|x)\Phi_\theta(x,b)\right). \tag{19}$$

In some cases computing the advantage function $A^*_\theta = Q^*_\theta - V^*_\theta$ is easier than computing the action-values. Luckily, $B(A^*_\theta(x,\cdot),\eta) = B(Q^*_\theta(x,\cdot),\eta)$, hence in order to compute π_θ it suffices to compute the advantage function.

4.2.5 Maximum entropy IRL

This method was recently proposed by Ziebart et al. (2008). It works by minimizing an empirical approximation to the Kullback-Leibler (KL) divergence between the distribution of trajectories generated by the expert's behavior and P_θ, a distribution of the form

$$P_\theta(\xi) = \frac{e^{\theta^T \Phi_\xi}}{Z(\theta)}, \quad Z(\theta) = \sum_{\xi'\in\Xi} e^{\theta^T \Phi_{\xi'}}.$$

Here Ξ is the set of all legal trajectories in the MDP considered, ξ, ξ' represent individual trajectories in Ξ, and for $\xi = (x_0, a_0, x_1, a_1, \ldots, x_{H-1})$, Φ_ξ is its feature count:

$$\Phi_\xi = \sum_{i=0}^{H-1} \phi(x_i, a_i).$$

Dissimilarity The Kullback-Leibler (KL) divergence of distributions P, Q defined over the countable domain Ω is

$$D_{KL}(P\|Q) = \sum_{\omega\in\Omega} P(\omega)\log\frac{P(\omega)}{Q(\omega)}.$$

Note that minimizing $D_{KL}(P\|Q)$ in Q is equivalent to minimizing $H(P,Q) = -\sum_{\omega\in\Omega} P(\omega)\log Q(\omega)$, the so-called cross-entropy of P and Q.

Let $Q = P_\theta$ and, as before, assume that we are given an i.i.d. sample $\mathcal{D} = \{\xi_1, \ldots, \xi_{N_{\text{traj}}}\}$ from P_E, the distribution over the trajectories underlying the expert's behavior. Thus, minimizing $D_{KL}(P_E\|P_\theta)$ in θ gives the best match in the family (P_θ) to P_E (as in density estimation). By our previous remark this is equivalent to minimizing $H(P_E, P_\theta)$, which can be approximated by

$$\begin{aligned} J(\theta,\mathcal{D}) &= -\frac{1}{N_{\text{traj}}}\sum_{i=1}^{N_{\text{traj}}} \log P_\theta(\xi_i) \\ &= -\theta^T \overline{\Phi}_E + \log Z(\theta). \end{aligned}$$

This defines the dissimilarity function to be minimized. Note that this is a convex dissimilarity function, hence gradient methods can be expected to perform well. It is somewhat disputable if this method could be called a method for inverse reinforcement learning problem since the dissimilarity does not use optimal policies.

Update step The negated gradient of the proposed dissimilarity function gives the update step of the algorithm:

$$\Delta_k = \bar{\Phi}_E - \sum_{\xi \in \Xi} P_{\theta_k}(\xi)\Phi_\xi .$$

Clearly, the above summation is intractable as the number of trajectories is in general infinite. One trick proposed by Ziebart et al. (2008) is to replace the above sum by a sum over the states. Unfortunately, in our case the state space is countably infinite, so this trick is not applicable. Instead, we will follow the approach of Charniak and Johnson (2005): we pick the paths from Ξ that have the largest probabilities and approximate the sum with the sum computed with the help of these paths. This approach is known as n-best reranking in the parsing literature, see e.g. Collins (2000).

A notable property of this approach is that when the dissimilarity is minimized (i.e., when $\Delta_k = 0$) the feature expectations are exactly matched under the distribution found. In fact, the equilibrium distribution is the one that has the largest entropy amongst all the distributions that satisfy this constraint on the feature expectations (Jaynes 1957), hence the name of the method.

4.3 Regularization

Regularization has already been mentioned in the context of the Max-Margin algorithm as a tool to facilitate model selection. Clearly, as such it can also be applied to other dissimilarity functions if one switches from minimizing J to minimizing the regularized dissimilarity J_λ defined by

$$J_\lambda(\mathcal{D};\theta) = J(\mathcal{D};\theta) + \lambda\|\theta\|_2^2,$$

where λ is some small positive constant. As all of the described methods (except the Projection algorithm) can be regarded as steepest descent methods, this regularization factor will appear as an additive term in the update steps:

$$\Delta_\lambda = \Delta - \lambda\theta.$$

An alternative to regularization with $\|\theta\|_2^2$ would be to regularize with $\|\theta\|_1$. Such a regularization could be useful if one expects θ to be sparse. However, this direction is not pursued any further here.

4.4 Relationships between the methods

The purpose of this section is to further explore the connections between the IRL methods discussed above.

First, one may notice that the Max-Margin method "interpolates" between MWAL and the Projection method in some sense. When setting $\ell = 0$ and $\lambda = 0$, the dissimilarity is the same as in the MWAL method, as $\sum_{x,a} r_\theta(x,a)\mu_\theta(x,a) = \theta^T \sum_{x,a} \phi(x,a)\mu_\theta(x,a) = \theta^T \bar{\Phi}_\theta$. Then the update of the Max-Margin algorithm becomes

$$\Delta_k = \bar{\Phi}_E - \bar{\Phi}_{r_{\theta_k}} .$$

We will refer to the method that uses these updates as the *Perceptron* algorithm, due to its analogy with the classical Perceptron algorithm (e.g., Freund and Schapire 1999). If we apply these updates multiplicatively, and assume that the signs of the optimal parameters are known, we get the MWAL algorithm. Furthermore, if we set $\lambda = 1$, $\ell = 0$, and use the special step sizes computed using (11) and (12), we get the update step of the Projection algorithm.

The MaxEnt method can also be related to the Perceptron method. While in the Perceptron algorithm, the updates are computed solely based on the difference between the feature expectations of the expert and that of the current optimal policy, MaxEnt proposes updates that are computed using a mixture over trajectories. If the probability assigned to the path underlying the optimal policy π_θ is large compared to the probability assigned to other paths then the update direction of the MaxEnt method will be close to that of the Perceptron update direction.

MaxEnt IRL is also related to the Policy Matching to some extent: they both employ an exponential family distribution to smooth their dissimilarity functions. The difference is that Policy Matching does smoothing at the action level ($\pi_\theta(a|x)$), while MaxEnt does smoothing on the trajectory level ($P_\theta(\xi)$). Ziebart et al. (2008) illustrate through an example that Policy Matching suffers from a so-called label bias (see Lafferty et al. 2001): if there are multiple optimal actions in one or more states, the behavior returned by matching the expert's policy will not necessarily reproduce the distribution over paths generated by the expert. However, in many applications (such as in parsing) reproducing the path distribution is not necessary to produce a good behavior.

Note that the methods differ substantially in the choice of the dissimilarity: Both the Projection method and MaxEnt aim to match feature expectations, while Policy Matching aims to match the expert's decisions at the level of the individual states. In the case of MWAL and Max-Margin the dissimilarity is specified by comparing the payoff for the expert and the payoff for the optimal policy, which is a somewhat less direct measure of how well the trajectories underlying the found policy match the observed trajectories. In fact, because of this approach may also suffer from the label bias problem: if the reward function found allows multiple optimal policies then there is no guarantee that the trajectories of the underlying optimal policy will match the observed trajectories. In the Max-Margin approach this problem is mitigated by the introduction of the loss function that encourages solutions that visit state-action pairs that are frequently visited by the expert.

5 Using inverse reinforcement learning to learn to parse

In this section we first present the common ideas underlying applying IRL techniques to parser training, followed by the description of the resulting parser training algorithms.

5.1 Common ideas

A crucial question in applying IRL to parsing is how to set up the features of the reward function. Although in theory the rewards could depend on the full partial parse tree, in order to facilitate comparison with standard PCFG training we chose (in line with Definition 1)

$$\phi_i(x,a) = \mathbb{I}(R_i = R^a), \quad i = 1, 2, \dots, n_R,$$

where $\{R_i\}_{i=1}^{n_R}$ is the list of all rules, i.e., the features are n_R dimensional and binary.

Trees from the treebank Λ are viewed as trajectories of an "expert". A single treebank tree will be denoted by $\tau_E \in \Lambda$. Although the trees do not allow us to know the exact sequence of "actions" taken by the expert (i.e., the ordering in which the human parser applied the production rules), luckily this information is not needed by the algorithms, since *any* admissible ordering of the rules giving rise to τ_E define the same tree and since the features depend only on what rules are taken and not on when these rules are taken. One simple approach then is to assume that the expert always chooses the leftmost unexpanded nonterminal and this is indeed the approach that we take. This yields a data set in the form of a series of state-action pairs, allowing us to apply IRL algorithms.

It might seem that this approach is problematic since there could be multiple optimal policies, leading to the label bias problem. However, the problem can be overcome if the set of admissible actions is restricted to those actions that expand the leftmost unexpanded nonterminal. This is exactly the approach followed here, for all the methods discussed previously. Note that this restriction does not change the optimal values, hence there is no loss of generality because of it.

In order to apply the presented IRL algorithms, we will need to compute feature expectations. The (approximate) computation of the expert's unconditional feature expectations $\bar{\Phi}_E$ is straightforward: it is the average feature count given the treebank trees:

$$(\bar{\Phi}_E)_i = \frac{1}{|\Lambda|} \sum_{\tau_E \in \Lambda} f(R_i, \tau_E), \quad i = 1, 2, \dots, n_R, \tag{20}$$

where $f(R, \tau_E)$ means the total count of rule R in the parse tree τ_E (cf. (1)). The feature expectation for the policy optimal w.r.t. the reward function r ($\bar{\Phi}_r$) can be computed for all MDPs $M^G_{y_{\tau_E}}$ as the feature count in the respective maximum scoring tree τ^*. The estimated feature expectations in the final MDP M^G is obtained by averaging over all treebank trees.

The computation of conditional feature expectations, $\Phi_r(x, a)$, is a bit more involved. Take a single MDP corresponding to some sentence and let x be a state in this MPD and $a \in \mathcal{A}(x)$. Let $c^a = (N, i, j)$. Let $\tau(x, a)$ be the terminal state when the optimal policy is followed in $M_{w_{ij},(N,i,j)}$ from the root of this MDP (cf. Sect. 3 for the definition of $M_{w_{ij},(N,i,j)}$). Then, up to an additive constant, $\Phi_r(x, a)$ equals[9] the feature counts in tree $\tau(x, a)$: $(\Phi_r(x, a))_k = f(R_k, \tau(x, a))$, $k = 1, \dots, n_R$. Note that the trees $\tau(x, a)$ (and hence the rule counts) are computed when computing the inside scores in Viterbi parsing.

5.2 Parser training algorithms

In this section we present the five parser training algorithms resulting from the respective IRL algorithms. A generic form of the algorithms is shown as Algorithm 1. The individual methods differ in the implementation of the **computeStepSize** and **computeUpdates** functions and the choice of the link function g. Here **computeStepSize**(k, Δ, Λ) computes the step size to apply for the next parameter update. Note that this computation is generally trivial, except for the Projection method and the adaptive step-size rule that we will study. The function **computeUpdates** should return the vector Δ_k and it will be given separately for each of the parser training methods.

In these updated we will assume that a Viterbi parser is used for obtaining a parse. Note, however, that all algorithms except Policy Matching can be implemented efficiently even if some other (efficient) parsing method is used. We will use the following subroutines in the description of the respective **computeUpdates** function:

[9] This additive constant vector will drop out when we will substitute into (19).

Algorithm 1 Generic incremental parser training algorithm

```
Input: corpus $\Lambda$, grammar $G_\theta$, iteration limit $k_{\max}$, update methods computeUpdate, computeStepSize, regularization coefficient $\lambda \geq 0$, link function $g$
for $k$ in $1, \ldots, k_{\max}$ do
    $\Delta \leftarrow 0$
    for $\tau_E \in \Lambda$ do
        $\Delta = \Delta + \mathbf{computeUpdate}(\tau_E, G_\theta, k) - \lambda\theta$
    end for
    $\alpha_k \leftarrow \mathbf{computeStepSize}(k, \Delta, \Lambda, G_\theta)$
    for $i$ in $1, \ldots, n_R$ do
        $\theta_i \leftarrow g(g^{-1}(\theta_i) + \alpha_k \frac{1}{|\Lambda|}\Delta_i)$
    end for
end for
```

insideScores(w_{1K}, G) returns the table of Viterbi inside scores for the sentence w_{1K} computed using the PCFG G.

viterbiParse(w_{1K}, V) returns the maximum scoring tree for sentence w_{1K} given the precomputed Viterbi scores V.

score(τ, G) returns the score of the tree τ in the grammar G.

maximumScoringTrees$(w_{1K}, G, n_{\text{trees}})$ returns a set of n_{trees} trees with highest total scores for sentence w_{1K} in the grammar G.

viterbiParseStartingWith$(x_{w_{1K}}, a, V)$ returns the maximum scoring subtree rooted at the constituent in x over the words in x, assuming that the first action taken is a, with respect to the Viterbi scores V.

state(τ_E, c) is a state of the parsing MDP corresponding to the partial parse tree that is obtained by removing all the descendants of c from τ_E.

isCorrect(a, τ_E) returns 1 if all constituents introduced by a are in τ_E, and returns 0 in all other cases. More formally, **isCorrect**(a, τ_E) returns 1 if and only if $a \in \mathcal{A}(\tau'_E)$ and $T(\tau'_E, a) \subset \tau_E$, where $\tau'_E = \mathbf{state}(\tau_E, c^a)$ and $T(\tau'_E, a)$ is the next state after taking action a in τ'_E.

From now on, we will use θ (as in the section on IRL) to denote the vector of rule scores $(\sigma(R_i))_{i=1}^{n_R}$. We will use the notation G_θ to denote the grammar G with rule scores given by θ.

5.2.1 The Projection algorithm

In the parser training algorithm derived from the Projection algorithm of Abbeel and Ng (2004), behaviors are represented with the total count of rules used during parsing the treebank. This way the distance between the treebank tree τ_E and the tree τ (see (9)) is directly related to the difference of specific rule counts in τ_E and in τ. In other words, the distance of two trees reflect the number of rules that appear in one tree, but not in the other tree. In order to use the Projection algorithm, α_k must be set to 1, and the regularization coefficient λ must also be set to 1. The subroutine **computeStepSize** computes the step sizes using (11) and (12). Note that computing the step sizes can be done efficiently if the feature expectations computed by subroutine **computeUpdate** are shared with subroutine **computeStepSize**.

The pseudocode of the resulting algorithm for computing the updates is displayed as Algorithm 2.

Algorithm 2 Update computation: Projection algorithm

```
Input: expert tree $\tau_E$, grammar $G_\theta$
Parameters: none
$V \leftarrow$ insideScores$(y_{\tau_E}, G_\theta)$
$\tau^* \leftarrow$ viterbiParse$(y_{\tau_E}, V)$
for $i$ in $1, \ldots, n_R$ do
    $\Delta_i \leftarrow f(R_i, \tau_E) - f(R_i, \tau^*)$
end for
return $\Delta$
```

Algorithm 3 Update computation: MWAL/Perceptron

```
Input: expert tree $\tau_E$, grammar $G_\theta$
Parameters: none
$V \leftarrow$ insideScores$(y_{\tau_E}, G_\theta)$
$\tau^* \leftarrow$ viterbiParse$(y_{\tau_E}, V)$
for $i$ in $1, \ldots, n_R$ do
    $\Delta_i \leftarrow -(f(R_i, \tau_E) - f(R_i, \tau^*))$
end for
return $\Delta$
```

5.2.2 MWAL/Perceptron with multiplicative updates

As previously shown in Sect. 4.2.3, this algorithm only differs from the Perceptron algorithm of Collins and Roark (2004) only because a multiplicative update is used. Note that since this algorithm assumes that all components of θ are strictly positive, while the scores associated with positive features are strictly negative (they are log-probabilities), we have to switch to using negative features to let the algorithm estimate the optimal negated scores. The resulting algorithm is shown as Algorithm 3. Note that when using the parameters found, we must also use the negated feature values when computing the parse for a tree. Alternatively, one may negate the weights found by the algorithm once the algorithm returns the final estimate. The link function must be set to $g = \exp$ when using this method.

5.2.3 Max-Margin Parsing

As this method emerges from the structured prediction community, it is no surprise that applying it to the parsing problem, we get a previously known method. The performance measure is essentially the same as that of the Max-Margin Parsing algorithm proposed by Taskar et al. (2004), but the optimization method is different as we follow Ratliff et al. (2007). According to Shalev-Shwartz et al. (2007) (see also the references therein) subgradient methods can be faster and more memory efficient than interior point or decomposition methods for Max-Margin problems. As a concrete example, exponentiated gradient descent in the dual variables can perform better than sequential minimal optimization (Bartlett et al. 2005; Globerson et al. 2007). Note that the MWAL algorithm can be regarded as implementing exponentiated gradient descent in the primal variables, so the above conclusion does not apply to it.

We chose $\ell(a) = -c_\ell f(R^a, \tau_E)$ $(a \in \mathcal{A})$ to be the loss function for the treebank tree τ_E. This loss thus encourages giving high reward to the frequently used rules. The resulting

Algorithm 4 Update computation: Max-Margin

```
Input: expert tree $\tau_E$, grammar $G_\theta$
Parameters: loss constant $c_\ell$
for $i = 1, \dots, n_R$ do
    $\theta'_i \leftarrow \theta_i - c_\ell f(R_i, \tau_E)$
end for
$V \leftarrow$ insideScores($y_{\tau_E}, G_{\theta'}$)
$\tau^* \leftarrow$ viterbiParse($y_{\tau_E}, V$)
for $i$ in $1, \dots, n_R$ do
    $\Delta_i \leftarrow (f(R_i, \tau_E) - f(R_i, \tau^*))$
end for
return $\Delta$
```

Algorithm 5 Update computation: Policy Matching

```
Input: expert tree $\tau_E$, grammar $G_\theta$
Parameters: temperature parameter $\eta$
$V \leftarrow$ insideScores($y_{\tau_E}, G_\theta$)
$\Delta \leftarrow 0$
for $c \in \tau_E$ do
    $x =$ state($\tau_E, c$)
    $\pi(\cdot|x) \leftarrow B(A^*_\theta(x, \cdot))$, where $A^*_\theta(x, \cdot)$ is computed from $V$ as in (5)
    for $a \in \mathcal{A}(x)$ do
        $\tau_\theta(x, a) \leftarrow$ viterbiParseStartingWith($x, a, V$)
        $\Phi(x, a) \leftarrow (f(R_i, \tau_\theta(x, a)))_{i=1}^{n_R}$
        $\partial\pi(a|x) \leftarrow \pi(a|x)\frac{1}{\eta}(\Phi(x, a) - \sum_{b\in\mathcal{A}(x)} \pi(b|x)\Phi(x, b))$
        $\Delta \leftarrow \Delta + \{$isCorrect$(a, \tau_E) - \pi(a|x)\}\partial\pi(a|x)$
    end for
end for
return $\Delta$
```

algorithm is shown as Algorithm 4. Note that the regularization term of the update is moved to the generic algorithm. As it was also noted beforehand, setting $c_\ell = 0$ yields a regularized version of the Perceptron algorithm of Collins and Roark (2004).

5.2.4 Policy matching

The pseudocode for this algorithm is shown as Algorithm 5. This method aims to match the actions for which **isCorrect**(a, τ_E) returns 1, i.e., the actions that introduce constituents that are in the parse tree τ_E. A smoothed near-optimal policy is computed using the optimal advantage function $A^*_\theta = A^*_{r_\theta}$. This function can be efficiently computed once the inside Viterbi scores V have been computed for all intervals $[i, j]$, $1 \le i \le j \le K$ and all nonterminal symbols. As noted earlier these scores are available without extra computation if we are using a Viterbi parser. The subtree $\tau_\theta(x, a)$ returned by **viterbiParseStartingWith**$(x_{w_{1K}}, a, V)$ can also be found easily by using V. In fact, all the interesting subtrees can be extracted in at most $\mathcal{O}(K^2 n_R)$ time, where K is the length of the sentence to parse.

Algorithm 6 Update computation: Maximum Entropy reranking

```
Input: expert tree $\tau_E$, grammar $G_\theta$
Parameters: number of parses $n_{\text{trees}}$
$\Delta \leftarrow 0$
$\mathcal{T}^* \leftarrow$ maximumScoringTrees$(y_{\tau_E}, G_\theta, n_{\text{trees}})$
for $\tau \in \mathcal{T}^*$ do
  $p(\tau) \leftarrow \exp(\textbf{score}(\tau, G_\theta))$
  for $i$ in $1, \ldots, n_R$ do
    $\Delta_i \leftarrow \Delta_i + p(\tau)(f(R_i, \tau_E) - f(R_i, \tau))$
  end for
end for
return $\frac{\Delta}{\sum_{\tau \in \mathcal{T}^*} p(\tau)}$
```

5.2.5 Maximum Entropy discriminative reranking

When applying the Maximum Entropy IRL method to parser training, we get an algorithm that is very close in nature to the reranking method of Charniak and Johnson (2005). The difference between the resulting algorithm shown as Algorithm 6 and the method proposed by Charniak and Johnson (2005) is the choice of features: we only use the simplest possible features, i.e., rule counts. Note that finding the n_{trees} best trees may need up to $\mathcal{O}(n_{\text{trees}} K^3 n_{nt}^3)$ time, where K is the length of the sentence to parse, as pointed out by Charniak and Johnson (2005).

6 Empirical evaluation

The aim of the empirical evaluation is multifold. First, we were interested in comparing the performance of some algorithms previously tested on parser training (Max-Margin Parsing, the Perceptron algorithm and Maximum Entropy discriminative reranking) with others that have not been tested on parser training before (the Projection algorithm, MWAL, Policy Matching). Second, we were interested in the sensitivity of the algorithms to the hyperparameters: we examined different step-size rules and settings of the regularization coefficient. Third, we were interested in the dependence of the results on the size of the training set. Finally, we were interested in comparing results obtained by following the standard practice of using a single hold-out set to measure performance with results if we use cross-validation.

We compared the algorithms on the Penn Treebank WSJ corpus. The settings that we used were the same as those used by Taskar et al. (2004), Titov and Henderson (2007) and Turian and Melamed (2006), i.e., we have trained and tested all of the methods on the sentences not longer than 15 words, and unless otherwise mentioned we used sections 2–21 of the corpus for training, section 22 for development, and section 23 for testing. The grammar that we used is a simple parent-annotated grammar extracted from the training set and transformed to Chomsky normal form.[10] This grammar is much simpler than the

[10] The parent-annotated grammar contains nonterminal symbols that are composed from the labels of a constituent and its parent constituent, i.e., NP^VP meaning a noun phrase which is an immediate child of a verb phrase. Such a grammar can be trivially extracted from a treebank. If a rule extracted from the corpus would have more than two non-terminals, one can always introduce some new non-terminals and break the rule into a number of rules that uses these non-terminals.

one used by Taskar et al. (2004) or those used in other more recent works. In fact, our grammar contains 639 nonterminal symbols only, which is approximately six times less than the number (3975) reported by Taskar et al. (2004). We trained the parameters for the binary rules only and used the scores from the default lexicon of the Berkeley Parser[11] which scores word-tag pairs with a smoothed estimate of $\log \frac{P(\text{tag}|\text{word})}{P(\text{tag})}$). We had a total of 3392 weights to train.

We have decided to stick with a simple grammar to make a thorough study of the various training algorithms feasible given the computational resources that were available to us. We have implemented the methods in Java, using code pieces from the Stanford Parser[12] and the Berkeley Parser, and run our experiments on the "condor" cluster of the Computer and Automation Research Institute of the Hungarian Sciences. The experiments took a total of ca. 30,000 hours of CPU time on PCs with 3 GHz processors and 2 gigabytes of RAM. 100 passes through the training data took approximately 8 hours of running time for the Max-Margin method, the Perceptron algorithm. MWAL and Policy Matching were approximately 10% slower. The same number of passes took 13 hours for MaxEnt. These are still extremely short training times as compared with the methods of Turian and Melamed (2006) (5 days) and Titov and Henderson (2007) (6 days). The training of the parser of Taskar et al. (2004) took several months, as mentioned by Turian and Melamed (2006). Note that to achieve state-of-the-art results significantly more complicated grammars are used which increases training time substantially. To give an example, to obtain state-of-the-art results (LP $= 91.4\%$, LR $= 90.4\%$, $F_1 = 90.9\%$ and EX $= 62.0\%$), the training time of the speed-optimized CRF-CFG training method of Finkel et al. (2008) was 2 hours *per pass* through the training data (using the same setup as ours), so 100 passes would have taken 200 hours with this method, roughly 20 times more than the training time needed for the grammar that we investigate. According to Finkel et al. (2008), to obtain state-of-the-art results both a good feature-set and a good training algorithm is needed. Here we decided to focus on the training algorithms, hoping that our results generalize when other models are used. However, the validation of this remains for future work.

We have used the standard ParsEval metrics (Black 1992) of labeled precision (LP), labeled recall (LR), the F_1 measure (the weighted harmonic mean of LP and LR), plus the ratio of exact matches (EX). Note that the original labels were used for computing LP and LR, instead of the parent-annotated symbols. LP, LR and EX are computed as follows: Pick a parse tree τ_E from the corpus and let w_{1K} be its yield. Let the parse tree obtained for w_{1K} by the method to be evaluated be τ^*. Then

$$\begin{aligned} \mathrm{LP}(\tau_E) &= \frac{|\tau^* \cap \tau_E|}{|\tau^*|}, \\ \mathrm{LR}(\tau_E) &= \frac{|\tau^* \cap \tau_E|}{|\tau_E|}, \\ \mathrm{EX}(\tau_E) &= \mathbb{I}\{\tau^* = \tau_E\}, \end{aligned}$$

and LP, LR, EX are obtained by computing the averages of the respective values over the corpus. We report some results as the percentage of improvement over some baseline (typically, the performance of the PCFG parser trained with Maximum Likelihood). For instance, if the parser trained with Maximum Likelihood reaches an F_1 measure of F_1^{ML} ($0 \le F_1^{ML} \le 1$) and

[11] http://nlp.cs.berkeley.edu/Main.html#Parsing (Petrov and Klein 2007).

[12] http://nlp.stanford.edu/software/lex-parser.shtml

the parser trained with Policy Matching reaches an F_1 measure of F_1^{PM} $(0 \leq F_1^{PM} \leq 1)$ then the error reduction in F_1 by PM relative to ML is given by

$$ER_{F_1}(PM) = 1 - \frac{1 - F_1^{PM}}{1 - F_1^{ML}} = \frac{F_1^{PM} - F_1^{ML}}{1 - F_1^{ML}},$$

which is then reported as a percentage.

We have run the algorithms for 100 passes, and measured performance on the training, development and test sets after all passes. After 100 passes, we selected the parser that attained a maximal F_1 score on the development set during the 100 passes.[13] Whenever we report a result for a specific hyperparameter setting (e.g., a specific step size or a specific regularization value), we mean the result that is given by the best parser selected this way. Note that increasing the number of passes did not improve the results any further for any of the algorithms. We have initialized the rule scores using the logarithm of the empirical estimate of the respective relative frequencies of the rules. The gradients have been normalized before adding the regularization factors to them or multiplying them with the step sizes except for the Projection algorithm. This helps with flat areas of the optimization surface, while (when decreasing step sizes are used) it does not hurt much close to the optimum. Max-Margin, MaxEnt and Policy Matching all have a single hyperparameter that needs to be selected. We have set the loss constant $c_\ell = 0.5$ for Max-Margin and the temperature parameter to $\eta = 0.1$ for Policy Matching because these values worked well in preliminary experiments. For MaxEnt reranking, we have used the value $n_{\text{trees}} = 50$ as proposed by Charniak and Johnson (2005). Unless otherwise stated, we set the regularization constant $\lambda = 0$ for all the methods.

6.1 The influence of the choice of the step sizes

First, we want to find out which step sizes are the most suitable for this particular problem. We tested constant step sizes, step sizes proportional to $\frac{1}{k}$, $\frac{1}{\sqrt{k}}$, and the iRprop step-size rule (Igel and Hüsken 2000). The proportional constant was obtained for each method by jointly optimizing over the number of passes and a number of possible values which were $\{0.01, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$. (In the subsequent experiments the constants found here were used.) The results for different step-size rules can be seen in Table 1 and on Fig. 3, parts (a) through (d). As the Projection algorithm uses fixed step-size parameters, we do not present the results for it on this table. The first thing to notice is that the two top performing methods are Max-Margin and Policy Matching, closely followed by MaxEnt. The standard selection of $\alpha_k \sim \frac{1}{k}$ leads to the least improvement in the parsing performance. We see that $\alpha_k \sim \frac{1}{\sqrt{k}}$ produces particularly good results for all of the methods, perhaps due to the fact that this step-size choice is known to improve robustness. Constant step sizes also perform well for similar reasons. Interestingly, MaxEnt seems to be very robust to the choice of step sizes: its performance is nearly identical for all examined step-size rules except iRprop.

Besides monitoring results on the test set, we also report results on the training set. Results on the training set help us detect overfitting, as well as to see a method's ability to adapt to the data. In fact, both a poor and overly good results on the training set are problematic, though by adding proper complexity regularization overly good results on the training set

[13] We do not report results for the development set. We note in passing that they are a bit lower than test set results but otherwise they correlate with them strongly.

Table 1 Results for different step-size choices. Abbreviations: ML = Maximum Likelihood, PE = Perceptron, MW = MWAL, ME = Maximum Entropy, MM = Max-Margin, PM = Policy Matching. The four parts of the table are labeled by the respective step sizes

	Test performance [%]				Training performance [%]			
	LP	LR	F_1	EX	LP	LR	F_1	EX
$\alpha_k \propto \frac{1}{k}$								
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.39	81.95	85.97	49.25	92.98	91.43	92.20	58.55
MW	90.43	82.69	86.38	50.41	92.91	91.92	92.41	58.77
ME	**91.30**	83.45	**87.20**	**53.73**	**93.40**	92.51	**92.81**	**61.23**
MM	90.57	82.41	86.30	50.74	92.98	91.78	92.38	59.23
PM	90.10	**83.77**	86.82	47.59	92.44	**92.87**	92.66	55.4
$\alpha_k = \alpha_0$								
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	89.53	82.26	85.74	50.24	92.70	91.96	92.32	58.24
MW	89.87	83.45	86.54	51.57	92.02	92.32	92.17	55.57
ME	91.35	83.53	87.27	53.26	93.83	92.62	92.81	60.41
MM	91.91	**84.47**	**88.03**	52.40	93.67	92.90	93.28	61.27
PM	**92.02**	84.25	87.96	**53.39**	**94.59**	**93.43**	**94.01**	**64.41**
$\alpha_k \propto \frac{1}{\sqrt{k}}$								
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.66	83.06	86.69	51.40	92.87	92.18	92.53	58.54
MW	90.18	83.19	86.54	50.91	92.63	92.23	92.43	58.52
ME	91.44	83.62	87.26	54.22	93.14	92.56	92.85	60.72
MM	91.64	**84.38**	87.86	52.07	93.63	93.02	93.33	61.65
PM	**92.13**	84.27	**88.03**	**54.22**	**94.51**	**93.22**	**93.86**	**63.94**
iRpop								
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	89.24	82.82	85.91	47.42	92.52	91.84	92.18	57.80
MW	89.16	81.82	85.33	48.42	92.42	91.50	91.96	57.23
ME	91.00	83.01	86.82	51.07	93.68	92.53	93.07	61.70
MM	91.09	83.34	87.04	53.23	94.11	92.86	93.48	62.36
PM	**91.75**	**83.92**	**87.66**	**54.22**	**95.05**	**93.64**	**94.34**	**66.85**

might be turned into good results on the test set. We see that in this respect the results obtained for the iRprop step-size rule are the most promising, followed by the results obtained with the step-size sequence $1/\sqrt{k}$. However, for simplicity we decided to run the further tests with the latter step-size sequence.

6.2 The influence of the regularization parameter

Next we examined how regularization influences the results. We expect that the methods which performed better on the training set will take the greatest advantage of regularization. The regularization constant is fixed in the Projection algorithm, so results are not shown for this algorithm. We report results with $\frac{1}{\sqrt{k}}$ step sizes. Although using step sizes proportional to $\frac{1}{\sqrt{k}}$ by itself has some regularization effect, using explicit regularization improves performance. The dependency of the F_1 measure on the choice of the regularization constant can be seen on Fig. 2. The curve for Perceptron is not shown to preserve clarity—qualitatively this curve is very similar to the curve obtained for MWAL. As expected, we see that regularization has a positive effect for the Policy Matching algorithm,

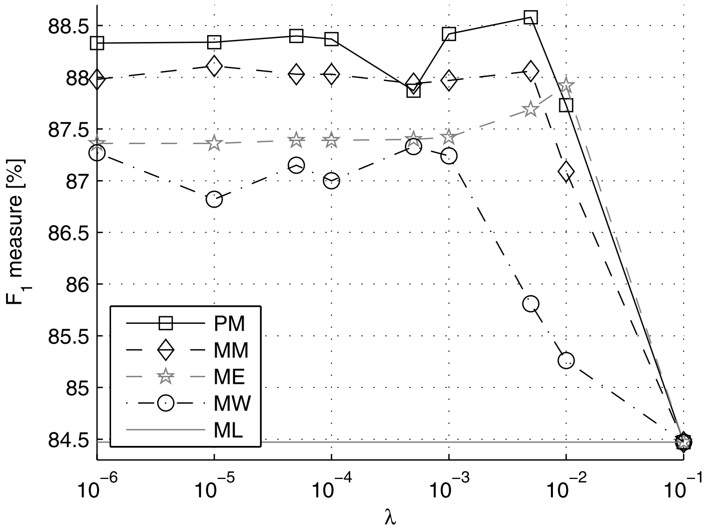


Fig. 2 F_1 measured on the test set vs. the regularization constant. The step size for pass k was set to $\frac{1}{\sqrt{k}}$. The graph for the Perceptron is not shown to maintain clarity

Table 2 Results for regularized parser training methods. The step size in pass k was set to $\frac{1}{\sqrt{k}}$ and the regularization parameters were optimized for each method. Abbreviations: ML = Maximum Likelihood, PE = Perceptron, MW = MWAL, ME = Maximum Entropy, MM = Max-Margin, PM = Policy Matching

	Test performance [%]				Training performance [%]			
	LP	LR	F_1	EX	LP	LR	F_1	EX
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PE	90.43	83.32	86.73	51.24	92.10	91.79	91.95	56.01
MW	90.54	83.77	87.02	52.73	92.06	91.97	92.01	56.87
ME	92.00	83.77	87.69	55.72	93.21	92.36	92.78	60.51
MM	92.17	84.40	88.11	52.90	93.83	92.86	93.34	61.66
PM	**92.86**	**84.68**	**88.58**	**55.80**	**94.36**	**93.05**	**93.70**	**64.05**

but it does not improve performance of the other methods significantly. The general tendency that can be observed in the graph is that the performance is roughly constant for small values of the regularization coefficient, and falls down quickly as the coefficient approaches 0.1. For Policy Matching and MaxEnt, we see that there is an interval of regularization values that improve performance. Table 2 and Fig. 3(e) show the performance of the parsers that were trained using regularization. To obtain these results, the regularization parameters were optimized on the development set by sweeping over a set of values for each individual method, while also optimizing for the number of passes. The set used was $\{10^{-6}, 5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}, \ldots, 10^{-1}\}$. (In subsequent experiments we used the regularization constants found in this step.)

6.3 The influence of the size of the training corpus

In the next set of experiments we measured how performance changes as a function of the size of the training set. For this experiment we used 1, 2, 5, 10 or 20 sections following sec-

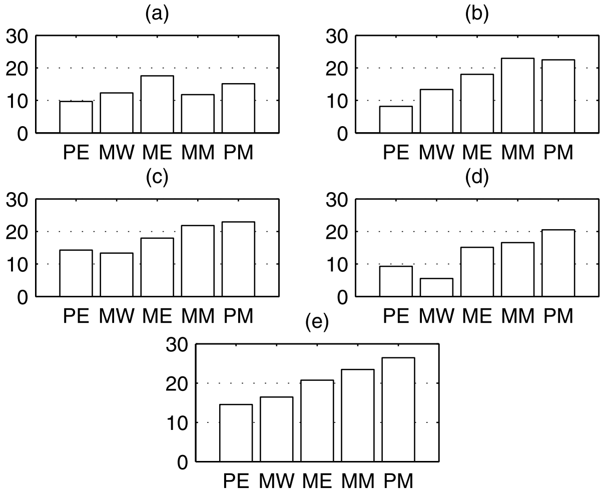


Fig. 3 Relative error reduction in F_1 over the ML method. The table shows test set results for various step-size rules and optimized regularization. Abbreviations: PE = Perceptron, MW = MWAL, ME = Maximum Entropy, MM = Max-Margin, PM = Policy Matching. The five parts show results for (**a**) $\frac{1}{k}$ step sizes; (**b**) constant step sizes; (**c**) $\frac{1}{\sqrt{k}}$ step sizes; (**d**) iRprop rule; and (**e**) $\frac{1}{\sqrt{k}}$ step sizes with regularization

tion 2 from the Penn Treebank WSJ corpus, and measured performance on section 23. The results are shown in Table 3. We see that for small training sets, the Perceptron and MWAL algorithms do a good job in fitting to the training examples, but generalize more poorly than the other three methods. As the size of the training set increases, Policy Matching gradually takes over them in the training set performance. On the test set, MaxEnt, Max-Margin and Policy Matching produce the best results, irrespective of the size of the training set. The Projection algorithm produces very poor results. Figure 4 shows the error reduction in F_1 over the baseline method (Maximum Likelihood) achieved by the different methods on the test set and the training set. Results for the Projection algorithm are not shown because this algorithm is not able to improve on the baseline.

The first observation is that as the size of the training set grows, error reduction increases on the test set, while it decreases on the training set. This is in line with the known fact that discriminative methods tend to work better for larger datasets (Ng and Jordan 2001). Note that there is a significant difference between the way the Perceptron and MWAL behave compared to the other methods: The former methods achieve the best error reduction on the *training set* initially, but they don't improve as much on the test set as the size of the training set is increased as the other methods.

6.4 Results with cross-validation

In this section we present results that were obtained with cross-validation. The motivation is to test the robustness of conclusions that can be drawn using the "standard" setup when performance is measured on a single hold-out set. For this reason, we performed 10-fold cross-validation on sections 2–22 of the corpus. Results are shown in Table 4. In these experiments the step size in pass k was set to $\frac{1}{\sqrt{k}}$. We provide results both with and without regularization. We have performed paired Kolmogorov–Smirnov-tests to see whether the measured differences are significant or not. Based on the results we see that MaxEnt, Max-Margin and Policy Matching perform significantly better than the Perceptron method and MWAL. The differences between the Perceptron method and MWAL, and those between MaxEnt, Max-Margin and Policy Matching are not significant. However, the effect of regularization on the performance of Policy Matching is statistically significant in the exact

Table 3 Results for training with training sets of different sizes. The step size at pass k was set to $\frac{1}{\sqrt{k}}$. Abbreviations: ML = Maximum Likelihood, PR = Projection, PE = Perceptron, MW = MWAL, MM = Max-Margin, ME = Maximum Entropy, PM = Policy Matching. Parts of the table are labeled by the respective number of sections that were used for training

	Test performance [%]				Training performance [%]			
	LP	LR	F_1	EX	LP	LR	F_1	EX
1 section								
ML	86.70	57.61	69.22	32.17	93.19	91.39	92.28	63.31
PR	79.02	56.37	65.80	23.88	85.37	88.96	90.07	40.81
PE	86.60	58.37	69.73	30.51	**97.15**	96.01	96.57	**78.19**
MW	86.46	58.52	69.80	31.67	97.02	**96.30**	**96.66**	76.93
ME	87.00	58.86	70.22	32.00	95.48	95.03	95.25	71.27
MM	**88.21**	**59.32**	**70.94**	**34.32**	96.82	95.69	96.25	74.63
PM	87.50	59.28	70.67	32.50	95.73	95.32	95.52	72.74
2 sections								
ML	87.65	63.95	73.95	**40.46**	93.02	90.24	91.61	59.79
PR	73.55	61.20	66.81	25.04	84.67	86.73	88.65	36.97
PE	87.19	65.06	74.52	36.65	95.19	94.28	94.73	68.09
MW	86.81	64.60	74.08	39.13	95.96	**95.31**	**95.63**	**70.85**
ME	88.04	**65.71**	75.25	39.96	93.48	93.72	93.60	63.68
MM	**88.81**	**65.71**	**75.53**	**40.46**	95.54	94.72	95.13	70.47
PM	88.34	65.32	75.10	39.30	**96.30**	94.56	95.42	69.34
5 sections								
ML	89.32	75.04	81.56	47.26	92.35	90.06	91.19	57.79
PR	80.97	69.98	75.07	30.84	85.10	83.56	86.80	32.38
PE	89.43	77.38	82.97	45.93	93.28	93.53	93.40	60.60
MW	90.08	76.95	83.00	46.76	94.26	93.29	93.77	64.34
ME	90.56	**77.75**	83.67	47.59	93.45	93.09	93.27	61.16
MM	90.49	77.34	83.40	47.59	94.47	93.56	94.01	64.61
PM	**91.39**	**77.75**	**84.02**	**49.58**	**95.24**	**93.89**	**94.56**	**65.80**
10 sections								
ML	88.64	78.27	83.13	47.59	92.30	89.99	91.13	58.50
PR	81.08	74.29	77.54	30.18	85.70	84.86	86.39	39.21
PE	89.23	80.98	84.90	49.58	92.96	93.14	93.05	59.85
MW	89.25	79.85	84.29	48.92	93.92	92.57	93.24	62.37
ME	90.49	81.24	85.61	51.40	93.63	93.00	93.32	62.52
MM	90.29	80.78	85.27	51.07	94.08	93.24	93.66	63.30
PM	**92.02**	**81.93**	**86.68**	**53.23**	**95.30**	**93.60**	**94.44**	**66.15**
20 sections								
ML	89.16	80.24	84.47	47.59	91.99	89.48	90.72	56.75
PR	80.58	73.38	76.81	34.16	85.79	82.37	84.12	39.28
PE	90.66	83.06	86.69	51.40	92.87	92.18	92.53	58.54
MW	90.18	83.19	86.54	50.91	92.63	92.23	92.43	58.52
ME	91.44	83.62	87.26	54.22	93.14	92.56	92.85	60.72
MM	91.64	**84.38**	87.86	52.07	93.63	93.02	93.33	61.65
PM	**92.13**	84.27	**88.03**	**54.22**	**94.51**	**93.22**	**93.86**	**63.94**

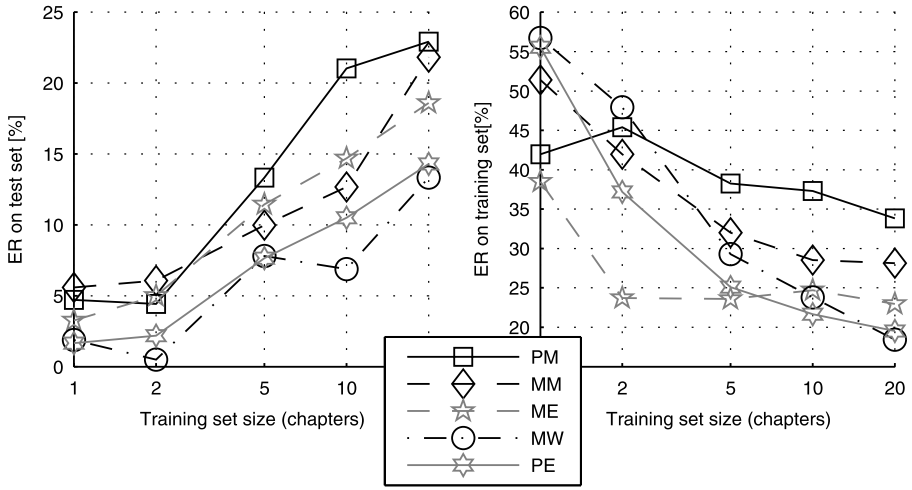
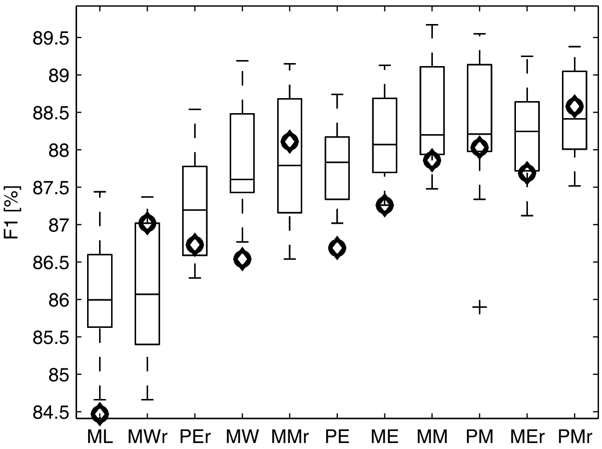


Fig. 4 Relative error reduction in F_1 over the ML method on the test set (*left hand side*) and on the training set (*right hand side*) as a function of the size of the training set. The step size used in the kth pass is $\frac{1}{\sqrt{k}}$ and no regularization is used. Results for Projection are not shown to maintain clarity

Table 4 Cross-validation results. Abbreviations: ML = Maximum Likelihood, PE = Perceptron, MW = MWAL, ME = Maximum Entropy, MM = Max-Margin, PM = Policy Matching. In the table μ denotes the estimated performance and σ is the estimated standard deviation. The upper part of the table shows results for the case when no regularization was used, while the lower part shows results for the case when regularization was turned on (the regularization constants were optimized for each method on the development set)

$\lambda = 0$	Test performance [%]							
	LP		LR		F_1		EX	
	μ	σ	μ	σ	μ	σ	μ	σ
ML	90.21	0.61	82.38	1.12	86.12	0.80	48.45	1.80
PE	90.74	0.58	85.16	1.18	87.86	**0.61**	49.68	2.09
MW	90.91	0.69	84.98	1.13	87.84	0.73	49.88	2.25
ME	91.44	0.63	85.33	**0.97**	88.28	0.63	51.94	**1.69**
MM	**91.89**	**0.57**	85.27	1.14	**88.45**	0.78	**52.05**	1.78
PM	91.24	1.39	**85.49**	1.20	88.26	1.07	48.94	5.42
$\lambda > 0$	LP		LR		F_1		EX	
	μ	σ	μ	σ	μ	σ	μ	σ
ML	90.21	0.61	82.38	1.12	86.12	0.80	48.45	**1.80**
PE	90.29	0.77	84.43	1.40	87.25	0.75	46.71	2.16
MW	89.28	0.71	83.14	1.61	86.09	0.95	45.90	2.46
ME	91.49	**0.51**	**85.07**	**1.03**	88.16	0.67	51.68	1.87
MM	91.56	0.67	84.55	1.22	87.91	0.87	50.59	2.47
PM	**92.23**	0.55	84.98	1.07	**88.46**	**0.64**	**52.80**	2.01

Fig. 5 Box plot of the F_1 error achieved by the different methods estimated by cross-validation. Abbreviations: ML = Maximum Likelihood, PE = Perceptron, MW = MWAL, ME = Maximum Entropy, MM = Max-Margin, PM = Policy Matching. The suffix "r" means "regularized". Boxes are ordered with respect to the estimated medians of the distributions. Circles mark test set results when methods are trained and tested in the standard setting



match ratio and the labeled precision, at the confidence level of 5%. Figure 5 shows a box plot that illustrates the distributions of F_1 measures achieved by the different methods.

We find that the performance measure on the single hold-out set are biased as compared with the results obtained using cross-validation: The averages of LR and F_1 computed measured on the single hold set are negatively biased, while EX is positively biased. Note that this bias does not cause any problems during the comparison of the methods if it is the same for the different methods, however, this does not hold. In particular, regularized Policy Matching and Max-Margin Parsing looks as if they had a definitely better performance than MaxEnt training when measured on the single hold-out set: When measured on the hold-out set, the relative error reduction in F_1 for Policy Matching is 26.46%, for Max-Margin it is 25.36%, while for MaxEnt it is 20.88%. However, the results of cross-validation predict no significant differences between these methods (see Fig. 5). Hence, we find that the current practice of measuring performance only on the last section may lead to false conclusions.

7 Conclusions

In this paper we proposed to reduce structured prediction problems, in particular, parser training problems to solving inverse reinforcement learning (IRL) problems. We have shown how IRL methods can lead to parser training methods. Although in this paper we concentrated on parser training based on PCFGs, we argued that the idea of the reductions carries through to other settings. As a result, the IRL problem can be a "least common denominator" of structured prediction problems and can provide an abstract, problem independent framework to study structured prediction problems.

Another contribution of the paper is a unified framework for presenting IRL algorithms. In particular, we have presented five IRL algorithms in the unified framework and then showed how they can be used to obtain various parser training methods. The unified framework suggests a few more possibilities: The link function could be chosen in various ways, or one could use stochastic gradient methods. Regularization could also be interpreted by averaging the weights found in the various iterations, possibly weighted with how well they perform on the development set. A further enhancement would be to use some voting scheme, see e.g. Carvalho and Cohen (2006).

The resulting algorithms were compared on the Penn Treebank WSJ corpus, both in a standard setting and with cross-validation. Our results suggest that the Maximum Entropy, the Max-Margin and the Policy Matching algorithms are the best performing methods, while the performance of MWAL, the Projection and the Perceptron methods are weaker. In terms of computation cost, amongst the best performing algorithms the subgradient implementation of Max-Margin training is the cheapest, followed by Policy Matching, which turned out to be ca. 10% more expensive. Our best parser was trained using regularized Policy Matching and achieved 88.58% F_1 accuracy on the test set. This means a 26.46% error reduction in F_1, as compared to our the baseline model trained with Maximum Likelihood. This is a significant error reduction as compared with the results of Taskar et al. (2004) whose training algorithm achieves only a 1.74% error reduction in the same measure. With the introduction of lexical features and using an auxiliary POS-tagger, they report a 9.4% error reduction over the baseline (however, they do not report results for their generative baseline model using this POS-tagger).[14] The large error reduction achieved here underlines that the choice of a good parser training method matters.

We find the connection between IRL and parser training especially fruitful in that it allows one to derive parser training algorithms from any IRL method. This connection suggests a number of further potential future enhancements. Further robustness might be gained by considering stochastic outcomes of the labelling decisions. By changing the way the rewards depend on the states (partial parses) new, more powerful models can be created that may lead to further performance improvements. The connection to RL could also be exploited by considering value function approximation methods that may result in significantly faster parsers and no loss of accuracy if one uses the approximate value functions together with appropriate search methods.

References

Abbeel, P., & Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML'04* (pp. 1–8).

Bakir, G. H., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., & Vishwanathan, S. V. N. (2007). *Predicting structured data (neural information processing)*. Cambridge: MIT Press.

Bartlett, P. L., Collins, M., Taskar, B., & McAllester, D. (2005). Exponentiated gradient algorithms for large-margin structured classification. In *Advances in neural information processing systems* (Vol. 17, pp. 113–120). Cambridge: MIT Press.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.

Black, E. (1992). Meeting of interest group on evaluation of broad-coverage parsers of English. In *LINGUIST list 3.587*. http://www.linguistlist.org/issues/3/3-587.html.

Boyd, S., El Ghaoui, L., Feron, E., & Balakrishnan, V. (1994). *Studies in applied mathematics: Vol. 15. Linear matrix inequalities in system and control theory*. Philadelphia: SIAM.

Carvalho, V. R., & Cohen, W. W. (2006). Single-pass online learning: performance, voting schemes and online feature selection. In *KDD'06* (pp. 548–553). New York: ACM.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge: Cambridge University Press.

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *ACL '05: Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 173–180). Morristown: Association for Computational Linguistics.

Collins, M. (1999). Head-driven statistical models for natural language processing. Ph.D. thesis, University of Pennsylvania.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *ICML'00* (pp. 175–182).

[14] The possible reasons for the low error reduction reported by Taskar et al. (2004) are discussed by Finkel et al. (2008).

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing* (pp. 1–8). Morristown: Association for Computational Linguistics.

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd annual meeting on association for computational linguistics* (pp. 111–118). Morristown: Association for Computational Linguistics.

Daumé III, H. (2006). *Practical structured learning techniques for natural language processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA.

Elliott, H., Derin, H., Cristi, R., & Geman, D. (1984). Application of the Gibbs distribution to image segmentation. In *Proc. 1984 int. conf. acoust., speech, signal processing, ICASSP'84* (pp. 32.5.1–32.5.4).

Finkel, J. R., Kleeman, A., & Manning, C. D. (2008). Efficient, feature-based, conditional random field parsing. In *ACL 08* (pp. 959–967). Morristown: Association for Computational Linguistics.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Globerson, A., Koo, T. Y., Carreras, X., & Collins, M. (2007). Exponentiated gradient algorithms for log-linear structured prediction. In *ICML '07: Proceedings of the 24th international conference on machine learning* (pp. 305–312). New York: ACM.

Igel, C., & Hüsken, M. (2000). Improving the Rprop learning algorithm. In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)* (pp. 115–121). San Diego: Academic Press.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review*, *106*(4), 620–630.

Klein, D., & Manning, C. D. (2003). A^* parsing: fast exact viterbi parse selection. In *NAACL '03: Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on human language technology* (pp. 40–47). Morristown: Association for Computational Linguistics.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th international conf. on machine learning* (pp. 282–289). San Mateo: Morgan Kaufmann.

Maes, F., Denoyer, L., & Gallinari, P. (2007). Sequence labeling with reinforcement learning and ranking algorithms. In *ECML* (pp. 648–657).

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge: MIT Press.

Neu, G., & Szepesvári, Cs. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Conference on uncertainty in artificial intelligence (UAI)* (pp. 295–302).

Ng, A., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *ICML-2000* (pp. 663–670).

Ng, A. Y., & Jordan, M. I. (2001). On discriminative vs. generative classifiers: a comparison of logistic regression and naive bayes. In *NIPS-14* (pp. 841–848).

Petrov, S., & Klein, D. (2007). Learning and inference for hierarchically split PCFGs. In *AAAI 2007 (nectar track)* (pp. 1663–1666).

Ratliff, N., Bagnell, J., & Zinkevich, M. (2006). Maximum margin planning. In. *ICML'06* (pp. 729–736).

Ratliff, N., Bagnell, J. D., & Zinkevich, M. (2007). Subgradient methods for structured prediction. In *Eleventh international conference on artificial intelligence and statistics (AIStats)* (pp. 2:380–387). (Online).

Rivas, E., & Eddy, S. R. (1999). A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, *285*(5), 2053–2068.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: primal estimated sub-GrAdient SOlver for SVM. In *ICML '07: Proceedings of the 24th international conference on machine learning* (pp. 807–814). New York: ACM.

Syed, U., & Schapire, R. (2008). A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems* (Vol. *20*, pp. 1449–1456). Cambridge: MIT Press.

Taskar, B., Klein, D., Collins, M., Koller, D., & Manning, C. (2004). Max-margin parsing. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)* (pp. 1–8).

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: a large margin approach. In *ICML '05: Proceedings of the 22nd international conference on machine learning* (pp. 896–903). New York: ACM.

Titov, I., & Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th annual meeting of the association of computational linguistics* (pp. 632–639). Prague: Association for Computational Linguistics.

Turian, J., & Melamed, I. D. (2006). Advances in discriminative parsing. In *ACL '06: Proceedings of the 21st international conference on computational linguistics and the 44th annual meeting of the ACL* (pp. 873–880). Morristown: Association for Computational Linguistics.

Warmuth, M. K., & Jagota, A. K. (1997). *Continuous and discrete-time nonlinear gradient descent: relative loss bounds and convergence* (Technical Report). Fifth International Symposium on Artificial Intelligence and Mathematics.

Ziebart, B., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI* (pp. 1433–1438).