

# Structured Prediction with Reinforcement Learning

Francis Maes<sup>1</sup> and Ludovic Denoyer<sup>1</sup> and Patrick Gallinari<sup>1</sup>

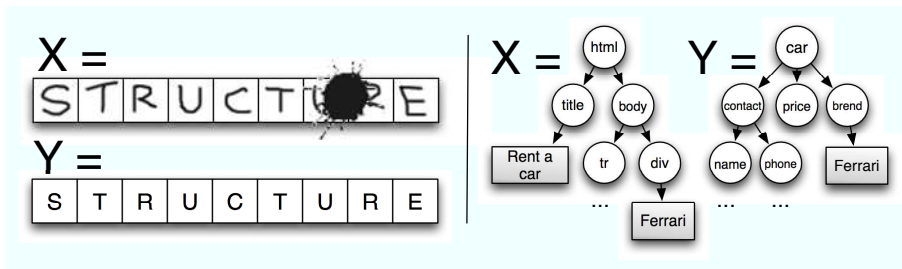
LIP6 - University Pierre et Marie Curie (Paris 6)

The date of receipt and acceptance will be inserted by the editor

**Abstract** We formalize the problem of Structured Prediction as a Reinforcement Learning task. We first define a *Structured Prediction Markov Decision Process* (SP-MDP), an instantiation of Markov Decision Processes for Structured Prediction and show that learning an optimal policy for this SP-MDP is equivalent to minimizing the empirical loss. This link between the supervised learning formulation of structured prediction and reinforcement learning (RL) allows us to use approximate RL methods for learning the policy. The proposed model makes weak assumptions both on the nature of the Structured Prediction problem and on the supervision process. It does not make any assumption on the decomposition of loss functions, on data encoding, or on the availability of optimal policies for training. It then allows us to cope with a large range of structured prediction problems. Besides, it scales well and can be used for solving both complex and large-scale real-world problems. We describe two series of experiments. The first one provides an analysis of RL on classical sequence prediction benchmarks and compare our approach with state-of-the-art SP algorithms. The second one introduces a tree transformation problem where most previous models fail. This is a complex instance of the general labeled tree mapping problem. We show that RL exploration is effective and leads to successful results on this challenging task. This is a clear confirmation that RL could be used for large size and complex structured prediction problems.

## 1 Introduction

We consider the prediction of structured objects, where each object may be described by a set of interdependent variables. In many different fields such as biology, natural language processing, image processing or chemistry,



**Fig. 1** Two examples of Structured Prediction. Left: sequence labeling, the input  $x$  is a sequence of handwritten characters and the output  $y$  is a sequence of labels identifying the recognized characters. Right: tree transformation,  $x$  is an HTML tree from the Web and  $y$  is an XML tree with additional semantic information.

data may be naturally described as structured objects like sequences, trees, lattices or graphs. Figure 1 illustrates the structured prediction problem. The left part of the figure is an example of sequence labeling where inputs are sequences of handwritten characters (*e.g.* gray-scale bitmaps) and outputs are sequences of labels identifying recognized characters. The right part of the figure illustrates a more complex problem where the goal is to map an input tree describing a document (an HTML page) onto an enriched structured representation of the same data (an XML tree with a specific Document Type Definition (DTD) for example).

Recently, structured prediction (SP) has witnessed a surge of interest in the ML community and several new ideas and models have thus emerged over the last few years. A key difficulty when dealing with SP is the combinatorial nature of the output space. In order to break this complexity, SP models usually make simplifying assumptions on the nature of the dependencies or on the loss functions used for training. Even then, many models are often restricted to problems with limited complexity and they do not scale, neither for inference nor for learning, with large datasets. A new vision of the SP process has been recently proposed where SP inference is considered as a sequential decision problem [5,7,6]. For this approach, the structured output is built incrementally: components are added one at a time to a current partial solution until a final solution is reached. Inference then consists in exploring a search space defined by states (partial solutions) and actions (choosing a component to be added to the current state) until a *complete* solution is built. After training, inference is usually performed in a greedy way, *e.g.* at each step, the best action according to a decision function is chosen. Going back to Figure 1, decisions for the sequence labeling example would simply consist in choosing the correct label for each input character. For the tree transformation example, dependencies are more complex and decisions may correspond to node creations, displacements or deletions.

As a follow up of this work [5,7,6], we investigate here the use of reinforcement learning algorithms for SP tasks. We introduce a new framework

called SP-MDP, which is a formulation of SP based on Markov Decision Processes. We show the equivalence between learning a policy in SP-MDP and minimizing the empirical risk of the corresponding SP problem. This equivalence between the SP supervised formulation and RL offers a principled way for handling SP problems using available RL methods. A second benefit of this approach is that RL only needs a weak supervision for learning. In this sense, it is more general than other incremental techniques. Besides, it inherits the nice properties of incremental methods and makes no assumption such as the decomposability of data representations or of the cost function, which are needed for some families of SP algorithms. The proposed framework allows us to express natural and efficient solutions to complex SP problems. It also scales well with both large datasets and large input or output dimensional spaces.

To summarize, the main contributions of this paper are threefold:

- *Connecting SP and RL:* We propose to formalize the Structured Prediction task in the Reinforcement Learning formalism, which allows us to use RL methods for solving SP problems.
- *Few assumptions:* RL methods only require a weak supervision and almost no assumptions concerning the problem itself. Our method can easily be applied to a large variety of complex SP tasks on which other state-of-the-art SP methods fail.
- *Experimental results:* We propose an experimental comparison of state-of-the-art SP methods and of the RL approach. RL is shown to behave as well as state-of-the-art SP models on sequence labeling tasks. We also introduce a complex instance of the labeled tree transformation problem. It is shown that the proposed method is able to deal with complex SP problems and to scale with large-scale collections.

The paper is structured as follows. First, we introduce the field of SP and give an overview of existing methods in Section 2. We then describe the SP-MDP formulation and show the equivalence between learning a policy in SP-MDP and solving the SP problem in Section 3. We discuss the benefits of using reinforcement learning algorithms for SP in Section 4. We present our formalism on the classical SP task of sequence labeling in order to illustrate the different concepts introduced, to analyze the behavior of the RL algorithms on simple SP tasks and to perform a comparison with state-of-the-art methods in Section 5. Finally, we demonstrate the power of RL on the challenging tree transformation SP task in Section 6.

## 2 Related Work

In this section, we give an overview of existing SP methods. We focus on *general purpose* SP models, which are not restricted to a specific application or structured data type.

### 2.1 Notations

We adopt a general definition of SP as a supervised learning problem, where the goal is to learn a mapping from inputs  $\mathbf{x} \in \mathcal{X}$  to outputs  $\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}$ . The outputs are discrete structured objects, such as sequences, trees or graphs.  $\mathcal{X}$  is the set of all possible inputs and  $\mathcal{Y}_{\mathbf{x}}$  is the set of candidate outputs for a given input  $\mathbf{x}$ . We denote by  $\mathcal{Y} = \cup_{\mathbf{x} \in \mathcal{X}} \mathcal{Y}_{\mathbf{x}}$  the full output space. The training set is denoted  $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i \in [1, n]}$ . Examples in  $D$  are *i.i.d.* samples from the unknown distribution  $\mathcal{D}_{\mathcal{X} \times \mathcal{Y}}$ . A loss function  $\Delta(\hat{\mathbf{y}}, \mathbf{y})$  quantifies the cost of predicting  $\hat{\mathbf{y}}$  instead of  $\mathbf{y}$ . The models are parameterized by a vector  $\theta \in \mathbb{R}^d$  and  $f_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$  denotes the prediction function corresponding to parameters  $\theta$ .

### 2.2 Global Models

One of the first ideas for SP has been to generalize existing classification methods to structured outputs. Let  $F(\mathbf{x}, \mathbf{y}; \theta)$  be a function that measures *how good the predicted output  $\mathbf{y}$  is, given input  $\mathbf{x}$* . Classification is considered here as a global optimization problem defined as:

$$f_{\theta}(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} F(\mathbf{x}, \mathbf{y}; \theta)$$

A common choice is to choose for  $F$  a linear function:

$$F(\mathbf{x}, \mathbf{y}; \theta) = \langle \theta, \phi(\mathbf{x}, \mathbf{y}) \rangle$$

where  $\langle \cdot, \cdot \rangle$  denotes the scalar product and  $\phi$  is an input-output *joint-description* function. Such a function jointly describes an input  $\mathbf{x}$  and a corresponding candidate output  $\mathbf{y}$  as a feature vector in  $\mathbb{R}^d$ . Global models usually require that the output feature set and the loss function both decompose. We briefly review some of the best-known models below.

The Structured Perceptron [4] is a generalization of the classical Perceptron, which was first used for part of speech tagging and chunking. Learning is performed by simulating inference and correcting the weight vector each time a wrong output is predicted.

Conditional Random Fields (CRFs) [16] use a log-linear probability function to model the conditional probability of an output  $\mathbf{y}$  given an input  $\mathbf{x}$ . CRFs are undirected graphical models where Markov assumptions are used in order to make inference tractable. The probability of an output can then be expressed as a product over output sub-structures.

Several methods extending the ideas of Support Vector Machines to SP have been proposed. SVM for Interdependent and Structured Output spaces (SVM-ISO, also known as SVM<sup>struct</sup>) [24] is a generalization of maximum margin classification to structured outputs. Maximum Margin Markov Network ( $M^3N$ ) [22] is another well-known SP model that relies on maximum

margin and graphical models. The two methods differ in the way they generalize the max-margin principle and in how they handle the potentially exponential number of constraints of the Quadratic Programming problem they solve.

Besides these key references, many other extensions of large margin or graphical models have also been proposed like [13] and [23]. As discussed in Section 4.1, global models for SP often suffer from scalability issues. This has led to the development of incremental models, which are able to deal with larger applications.

### 2.3 Incremental Models

Instead of modeling *what a good prediction looks like*, incremental approaches directly model *how to build the good prediction*. This simple idea thus suggests integrating learning and searching into a sequential prediction process. Figure 2 gives an example of such a decision-making problem. Inference starts with an initial default output. Each decision corresponds to an elementary modification of the output being predicted. States contain *partial outputs* and final states contain *complete outputs*.

Incremental Parsing [5] is one of the first models using the idea of incremental prediction. This model was introduced in the context of natural language parsing, where inputs are sequences of words and outputs are parse trees. Incremental Parsing is built around a Perceptron that assigns ranking scores to partial outputs. The inference is performed greedily by taking decisions that maximize the ranking score. Each such decision adds a set of nodes to the currently built parsing tree. Learning is performed by repeating the following process: run the inference procedure until a wrong decision happens, stop inference and make an elementary correction using the Perceptron rule. In order to detect wrong decisions, Incremental Parsing assumes that the Optimal Learning Trajectories (OLTs) are known for all learning examples. An OLT is a sequence of actions that, given an input, leads from the initial state to the correct output.

Incremental prediction was popularized by LaSO [7], which is probably the first general Incremental SP model. LaSO relies on a beam-search procedure. The selection of partial outputs in the beam-search is computed by using a Perceptron. LaSO makes the same OLT assumptions as Incremental Parsing. An error is said to occur when the current beam does not cover the OLT anymore. Learning repeats the following steps until convergence: pick an example, run the inference procedure until an error occurs, correct the Perceptron *w.r.t.* the current error, correct the error – reinsert the OLT into the beam – and continue inference.

Searn [6] is another general Incremental SP model developed later. In Searn, the decision maker is modeled by a classifier (*e.g.* Support Vector Machines or Decision Trees). Searn assumes that for each learning example, we know an Optimal Learning Policy (OLP). The OLP is a procedure that

knows the best decision to perform for *any state* of the prediction space. Note that this is a stronger assumption than the one in LaSO. Searn uses an iterative batch-learning approach. At each Searn iteration, a mixture of the optimal decision maker and the learned decision maker is used to perform inference on all learning examples. For each visited state, one classification example is created. At the end of the iteration, a new classifier is trained using these new classification examples. This is repeated until convergence. Searn has shown to be efficient on numerous tasks and is considered as a general-purpose state-of-the-art SP model.

Incremental models can cope with non-additive loss functions. They do not rely on dynamic programming and are thus probably better adapted to complex and large-scale SP problems than global models are.

### 3 Structured Prediction with Markov Decision Processes

In this section, we introduce a new formulation of SP based on Markov Decision Processes. SP-MDPs are MDPs that model the inference process of a structured prediction. As usual in SP, two problems have to be solved:

- **Inference:** Given the parameters  $\theta$  and input  $\mathbf{x}$ , inference consists in selecting an output  $\hat{\mathbf{y}}$  among all candidates  $\mathcal{Y}_{\mathbf{x}}$ . The predicted output  $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$  should have a low loss value  $\Delta(\hat{\mathbf{y}}, \mathbf{y})$ .
- **Training:** Given the set of examples  $D$  and the loss function  $\Delta$ , training corresponds to minimizing the *expected risk* defined as follows:

$$\theta^* = \operatorname{argmin}_{\theta \in \mathbb{R}^d} E_{\mathbf{x}, \mathbf{y} \sim \mathcal{D}_{\mathcal{X} \times \mathcal{Y}}} \{ \Delta(f_{\theta}(\mathbf{x}), \mathbf{y}) \}$$

Since the distribution  $\mathcal{D}_{\mathcal{X} \times \mathcal{Y}}$  is unknown, the *expected risk* cannot be computed and one usually minimizes the *empirical risk*<sup>1</sup>:

$$\theta^* = \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} \Delta(\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

The main contribution of this section is to show that learning the optimal policy in an SP-MDP is equivalent to solving the corresponding SP problem, *i.e.* minimizing the empirical risk.

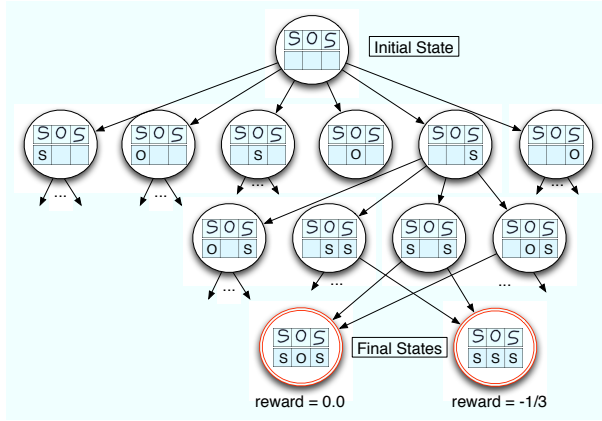
#### 3.1 SP Markov Decision Process

We adopt the formalism of deterministic MDPs  $(\mathcal{S}, \mathcal{A}, T, r)$  where  $\mathcal{S}$  is the state-space,  $\mathcal{A}$  is the set of possible actions,  $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$  is the transition function between states and  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function.

An SP-MDP is a deterministic MDP that models inference for a given SP task. SP-MDPs are illustrated in Figure 2 and defined formally below:

---

<sup>1</sup> or some regularized empirical risk.



**Fig. 2** Sequence labeling SP-MDP. Circles are states and links are transitions. Each state contains the input and a partial output. Here, the input is a sequence of three *black-and-white* bitmaps representing handwritten digits. Partial outputs are partially recognized sequences of digits. The bottom double circled states are final states containing complete outputs.

- *States.* Each state of an SP-MDP contains both an input  $\mathbf{x}$  and a partial output  $\bar{\mathbf{y}}$ . Let  $\bar{\mathcal{Y}}$  be the set of all possible partial outputs<sup>2</sup>. The set of states of an SP-MDP is then  $\mathcal{S} = \mathcal{X} \times \bar{\mathcal{Y}}$ . There is one initial state per possible input  $\mathbf{x}$ :  $\mathbf{s}^{initial}(\mathbf{x}) = (\mathbf{x}, \bar{\mathbf{y}}_\epsilon)$  where  $\bar{\mathbf{y}}_\epsilon \in \bar{\mathcal{Y}}$  is the initial empty solution. A set of examples  $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i \in [1, n]}$  can thus be mapped to a set of corresponding SP-MDP initial states  $\{\mathbf{s}^{initial}(\mathbf{x}^{(i)})\}_{i \in [1, n]}$ . Final states contain the complete outputs that can be returned to the user.
- *Actions.* Actions of SP-MDPs concern elementary modifications of the partial output  $\bar{\mathbf{y}}$ . Those elementary modifications are specific to the SP task. For example, for sequence labeling tasks, partial outputs are partially labeled sequences and an elementary modification might be the addition of a single label prediction to the current partial output. We denote  $\mathcal{A}_{\mathbf{s}} \subset \mathcal{A}$  the set of actions available in state  $\mathbf{s}$ . Action sets will be described in details for all the experiments in Sections 5 and 6.
- *Transitions.* SP-MDP transitions are deterministic and replace the current partial output by the transformed partial output. Transitions do not change the current input:

$$T((\mathbf{x}, \bar{\mathbf{y}}), \mathbf{a}) = (\mathbf{x}, \mathbf{a}(\bar{\mathbf{y}}))$$

where  $\mathbf{a}(\bar{\mathbf{y}})$  denotes the partial output modified by action  $\mathbf{a}$ .

- *Rewards.* In SP, the aim is to predict outputs that are as similar as possible to the correct outputs *w.r.t.* the loss function  $\Delta$ . When dealing with MDPs, the goal is expressed through rewards that should be maximized. In order to

<sup>2</sup> Note that complete outputs are included into the set of partial outputs, *i.e.*  $\mathcal{Y} \subset \bar{\mathcal{Y}}$

relate learning in MDPs with SP, the loss function  $\Delta$  will be incorporated into the rewards. This can be done in several different ways. We explore here two types of rewards:

- **Per-episode rewards** The per-episode reward function of an SP-MDP is defined as follows:

$$r_{episode}(\mathbf{s} = (\mathbf{x}, \bar{\mathbf{y}}), \mathbf{a}) = \begin{cases} -\Delta(\hat{\mathbf{y}} = \mathbf{a}(\bar{\mathbf{y}}), \mathbf{y}), & \text{if } T(\mathbf{s}, \mathbf{a}) \text{ is a final state} \\ 0, & \text{for all other states} \end{cases}$$

The reward is null for all actions, but those terminating the inference process. The negative loss of the predicted output is then given as reward. When using per-episode rewards, the credit assignment problem is maximally difficult, *i.e.* learning algorithms have to dispatch the reward over whole sequences of actions.

- **Per-decision rewards** It is usual in RL to use richer reward signals in order to help the learning and exploration processes. We have used here an alternative to per-episode rewards, which consists in providing rewards after each decision. It is therefore assumed that the loss function can be generalized to partial predicted outputs:  $\Delta : \bar{\mathcal{Y}} \times \mathcal{Y} \rightarrow \mathbb{R}$ . The per-decision reward function is then defined as follows:

$$r_{decision}(\mathbf{s} = (\mathbf{x}, \bar{\mathbf{y}}), \mathbf{a}) = -(\Delta(\mathbf{a}(\bar{\mathbf{y}}), \mathbf{y}) - \Delta(\bar{\mathbf{y}}, \mathbf{y}))$$

*i.e.* rewards given after each step correspond to the difference of loss induced by the action. Note the RL formulation of SP naturally leads to the definition and use of such partial losses. When it is possible to measure a loss on the final outputs, the same loss can usually be used on partial outputs too. This is the case for all the experiments described in this paper and this can be generalized to most SP problems.

In the following, we assume that the number of steps  $T$  required to construct a complete output  $\hat{\mathbf{y}}$  is bounded and only depends on the current input  $\mathbf{x}$ . This means that for a given  $\mathbf{x}$ , all the MDP trajectories will have the same length  $T$ . For example, in sequence labeling, a sequence of length  $n$  requires  $T = n$  steps to be constructed. Similarly, for labeled tree transformation, the number of steps will only depend on the input tree. We will denote in the following  $T^{(i)}$  the number of steps required to process the input  $\mathbf{x}^{(i)}$ .

### 3.2 Reinforcement Learning and Structured Prediction

When dealing with MDPs, the aim is to learn a decision maker, called a *policy*. We focus on deterministic policies  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  which are functions that map states to actions.

Given an initial state  $\mathbf{s}$  and a policy  $\pi$ , we consider the total reward optimality criterion. This is the most natural reward in our context. Since

the horizon is bounded, for any input  $\mathbf{s}$ , the total reward can always be expressed as a finite sum<sup>3</sup>:

$$\eta(\pi, \mathbf{s}) = \sum_{t=1}^T r(\mathbf{s}_t, \pi(\mathbf{s}_t)) \mid \pi, \mathbf{s}_1 = \mathbf{s}$$

The aim of RL algorithms is to find an optimal policy, i.e. a policy that maximizes the expectation of the total reward according to an initial state distribution  $\mathcal{D}_S$ <sup>4</sup> :

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_S} \{\eta(\pi, \mathbf{s})\}$$

In practice, most RL algorithms make use of policies  $\pi_\theta$  defined by a set of parameters  $\theta \in \mathbb{R}^d$ . The aim is then to find parameters  $\theta^*$  maximizing the expectation of total reward:

$$\theta^* = \operatorname{argmax}_{\theta \in \mathbb{R}^d} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_S} \{\eta(\pi_\theta, \mathbf{s})\}$$

### 3.3 Equivalence with SP

We now discuss the reward function of SP-MDPs and show that maximizing the expectation of total reward is equivalent to minimizing the empirical SP risk. This will provide a link between the supervised learning formulation of SP and RL. In the following,  $\mathbf{s}_t$  is the state at time-step  $t$  and  $\bar{\mathbf{y}}_t$  is the corresponding partial output. The predicted output corresponds to the final state  $\mathbf{s}_{T+1}$  and is denoted  $\hat{\mathbf{y}} = \bar{\mathbf{y}}_{T+1}$ .

• *Per-episode reward.* The total reward of one trajectory in SP-MDP is the negative loss of the predicted output:

$$\begin{aligned} \eta(\pi, \mathbf{s}) &= r(\mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1) + \dots + r(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}) + r(\mathbf{s}_T, \mathbf{a}_T) \\ &= 0 + \dots + 0 - \Delta(\bar{\mathbf{y}}_{T+1}, \mathbf{y}) = -\Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y}) \end{aligned}$$

Let us consider the distribution  $\mathcal{D}_S^D$  that uniformly picks examples from  $D$  to build initial states  $\mathbf{s}^{initial}(\mathbf{x})$ . When training an RL algorithm with the  $\mathcal{D}_S^D$  initial state distribution, we are looking for the optimal policy:

$$\begin{aligned} \pi_\theta^* &= \operatorname{argmax}_{\theta \in \mathbb{R}^d} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_S^D} \{\eta(\pi_\theta, \mathbf{s})\} \\ &= \operatorname{argmin}_{\theta \in \mathbb{R}^d} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim D} \{\Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}), \mathbf{y})\} \\ &= \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} \Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) \end{aligned}$$

<sup>3</sup> Assuming reward-free absorbing final states

<sup>4</sup> We introduce the distribution of initial states in order to properly define the maximization problem tackled by *approximated* RL algorithms.

The *optimal policy* with the initial state distribution  $\mathcal{D}_S^D$  is thus *the policy that minimizes the empirical risk*.

- *Per-decision reward.* When using per-decision rewards, the total reward of one trajectory is:

$$\begin{aligned}\eta(\pi, \mathbf{s}) &= r(\mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1) + \dots + r(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}) + r(\mathbf{s}_T, \mathbf{a}_T) \\ &= -(\Delta(\bar{\mathbf{y}}_2, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_1, \mathbf{y})) - (\Delta(\bar{\mathbf{y}}_3, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_2, \mathbf{y})) \\ &\quad \dots - (\Delta(\bar{\mathbf{y}}_{T+1}, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_T, \mathbf{y})) = -(\Delta(\hat{\mathbf{y}}, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_1, \mathbf{y}))\end{aligned}$$

Since the initial output is  $\bar{\mathbf{y}}_\epsilon$ , we have:  $\eta(\pi, \mathbf{s}) = -(\Delta(\hat{\mathbf{y}}, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_\epsilon, \mathbf{y}))$ . The total reward of one trajectory with per-decision rewards is the negative loss of the predicted output plus a constant value. Minimizing the expected total reward can thus be reformulated as follows:

$$\begin{aligned}\pi_\theta^* &= \operatorname{argmax}_{\theta \in \mathbb{R}^d} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_S^D} \{\eta(\pi_\theta, \mathbf{s})\} = \operatorname{argmin}_{\theta \in \mathbb{R}^d} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim D} \{\Delta(\hat{\mathbf{y}}, \mathbf{y}) - \Delta(\bar{\mathbf{y}}_\epsilon, \mathbf{y})\} \\ &= \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} \Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) - \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} \Delta(\bar{\mathbf{y}}_\epsilon, \mathbf{y}^{(i)}) \\ &= \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} \Delta(\hat{\mathbf{y}} = f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})\end{aligned}$$

With both rewards, the *expectation of total reward maximization* problem is thus equivalent to the *SP empirical risk minimization* problem.

Note that the correct outputs are required to compute rewards. These outputs are available for training but not for testing on new inputs. Rewards can then be computed only for the subset of the SP-MDP corresponding to states reachable from a initial training state  $s^{initial}(\mathbf{x}^{(i)})$ . SP-MDP are thus different from traditional RL problems, where the reward function is known anytime. From the point of view of RL, this special setting has two main consequences: a policy can only be trained on a subset of the SP-MDP and approximated policies are needed in order to generalize to the whole state-action space.

### 3.4 Approximate RL algorithms

The equivalence between RL and SP described previously makes it possible to use a large range of RL algorithms to perform SP. In our experiments, we have applied algorithms which are representative of two families: value-based RL and policy gradient RL.

- *Approximated Value-based RL.* A usual way to define a policy is to introduce an *action-value* function  $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  that assigns scores to candidate actions  $\mathbf{a}$  in a given state  $\mathbf{s}$ . This allows us to define the *greedy* policy, which always chooses actions that maximize  $Q$ :

$$\pi_Q^{greedy}(\mathbf{s}) = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}_s} Q(\mathbf{s}, \mathbf{a})$$

Approximation for large MDPs such as ours is often introduced via a linear function:

$$Q_{\theta}(\mathbf{s}, \mathbf{a}) \stackrel{\text{approx}}{=} \langle \theta, \phi(\mathbf{s}, \mathbf{a}) \rangle$$

where  $\phi(\mathbf{s}, \mathbf{a})$  is a joint state-action feature description function and  $\theta$  are the parameters of the approximated value function. In order to illustrate the use of value-based RL, we use the approximated SARSA(0) algorithm [21].

Note that SARSA, as most value-based RL algorithms, maximizes a discounted reward function:

$$\eta(\pi, \mathbf{s}) = \sum_{t=1}^{\infty} \gamma^t r(\mathbf{s}_t, \pi(\mathbf{s}_t))$$

where the choice of the discount factor  $\gamma$  leads to a continuous range of problems of increasing complexity, from maximizing the immediate reward ( $\gamma = 0$ ) to maximizing the total reward ( $\gamma \rightarrow 1$ ). In our case, since the horizon is bounded for each example, we can use  $\gamma = 1$ , which corresponds to a total reward. However, in the experiments, we have also performed tests with smaller discount values. This point is discussed in Sections 5 and 6.

• *Policy Gradient RL.* Policy gradient algorithms directly solve RL problems, without using value functions. These algorithms make use of stochastic policies and estimate the gradient  $\nabla_{\theta} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_S} \{\eta(\pi, \mathbf{s})\}$  through simulation. In our experiments, we used the OLPOMDP algorithm [1] with log-linear policies. This algorithm maximizes the expected reward-per-step in infinite horizon problems. After each decision step, it updates the current estimate of the expected reward-per-step gradient and performs a stochastic ascent update on the parameters  $\theta$ . In order to apply OLPOMDP on our decision problems, we adopt the same approach as [12]: we create an infinite horizon problem by connecting each final state of an example  $i \in [1, n]$  to the initial state of the next example. On this new problem, the limit at infinity of the expected reward-per-step is equal to the expected reward-per-step for one pass on the training set  $D$ :

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T r(s_t, \pi(s_t)) = \frac{1}{T_D} \sum_{t=1}^{T_D} r(s_t, \pi(s_t))$$

where  $T_D = \sum_{i=1}^n T^{(i)}$  is the total number of steps required to process all the examples of  $D$ . The average reward-per-step for one pass on the training set is equal to the SP empirical risk multiplied by a constant factor  $\frac{n}{T_D}$ :

$$\begin{aligned} \frac{1}{T_D} \sum_{t=1}^{T_D} r(s_t, \pi(s_t)) &= \frac{1}{\frac{T_D}{n} n} \sum_{i=1}^n \sum_{t=1}^{T^{(i)}} r(\mathbf{s}_t^{(i)}, \pi(\mathbf{s}_t^{(i)})) \text{ with } \mathbf{s}_1^{(i)} = \mathbf{s}^{\text{initial}}(\mathbf{x}^{(i)}) \\ &= \frac{n}{T_D} \left( \frac{1}{n} \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D} -\Delta(\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) \right) \end{aligned}$$

Maximizing the *expected reward-per-step* is thus equivalent to the *SP empirical risk* minimization. Once OLPOMDP learning is finished, as for SARSA, we use the deterministic policy:

$$\pi(\mathbf{s}) = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}_{\mathbf{s}}} \langle \theta, \phi(\mathbf{s}, \mathbf{a}) \rangle$$

### 3.5 Inference with an SP policy

Once a policy has been learned in an SP-MDP, it can be used for inference on any new incoming example. In order to make a prediction, we start with the initial partial output and choose the actions of the learned policy until reaching a final state. In order to choose the action in state  $\mathbf{s}$ , we evaluate all available actions in  $\mathcal{A}_{\mathbf{s}}$  and take the one that maximizes  $\langle \theta, \phi(\mathbf{s}, \mathbf{a}) \rangle$ . The final state then contains the predicted output. The complexity of the inference process is  $\mathcal{O}(T.a)$  where  $T$  is the depth of the SP-MDP and  $a$  is the mean number of available actions:  $\mathbb{E}\{\operatorname{card}(\mathcal{A}_{\mathbf{s}})\}$ .

## 4 Discussion

Having introduced the SP-MDP model and its use with RL algorithms, we will now compare this model to state-of-the-art methods. We first provide a comparison of incremental versus global models and then discuss in more details the specificities of the proposed approach *w.r.t.* two reference incremental methods: LaSO and Searn.

### 4.1 Incremental Models versus Global Models

All global models assume that the following *argmax* problem can be solved efficiently:

$$f_{\theta}(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} F(\mathbf{x}, \mathbf{y}; \theta)$$

This step, which is most of the time involved in both learning and inference, is usually handled with dynamic programming techniques. This has two major drawbacks. First, dynamic programming requires strong independence assumptions, which are often unrealistic for real-world data. For example, in sequence labeling, it is often assumed that a label only interacts with the previous and next labels. This strategy does not allow us to cope with complex output dependencies. Second, even with such independence assumptions, dynamic programming algorithms may have a prohibitive complexity. For example, when predicting trees, the best dynamic programming algorithms have a cubic complexity in the number of leaf nodes [20]. This limits the use of such methods to small trees, *e.g.* less than 50 nodes. More generally, the *argmax* problem prohibits the use of global methods on large-scale problems.

The incremental models aim at solving this dynamic programming step using greedy inference. They can be applied both for large-size problems and for complex dependencies. On the other hand, when facing local ambiguities, greedy inference may lead to sub-optimal solutions. In such cases, when tractable, global models have more chances to find the best compromises.

#### 4.2 SP-MDP versus other Incremental Models

The main difference between the proposed RL approach and the LaSO and Searn algorithms concerns the supervision information the algorithms rely on. LaSO and Searn are respectively based on the *Optimal Learning Trajectory* (OLT) and *Optimal Learning Policy* (OLP) assumptions (see Section 2.3), while SP-MDP is not limited by these assumptions.

In some simple SP tasks, the OLTs and the OLP can be computed trivially in  $\mathcal{O}(1)$ . This is the case for example in sequence labeling, where, whatever the current state is, the best decision is to select a correct label. In some other tasks, such as the tree transformation described in Section 6, OLTs and OLPs computation is a non-trivial or even intractable combinatorial search problem. In particular, computing the OLP means solving the following equation:

$$\pi^*(\mathbf{s}) = \operatorname{argmin}_{\mathbf{a} \in \mathcal{A}_s} \left\{ \min_{\hat{\mathbf{y}} \in \mathcal{U}(T(\mathbf{s}, \mathbf{a}))} \Delta(\hat{\mathbf{y}}, \mathbf{y}) \right\}$$

where  $\mathcal{U}(\mathbf{s}) \subset \mathcal{Y}$  is the set of reachable solutions when starting from state  $\mathbf{s}$ . In cases where the OLP is not trivially computed, a possibility is to directly solve the search problem given above. This is the solution proposed in [6] where the authors use a greedy beam-search algorithm for finding an approximate OLP for automatic summarization. However, this combinatorial problem may be too complex.

When using SP-MDP with classical RL algorithms like SARSA or OLPOMDP, we neither need to provide an OLP nor OLTs. In the following sections, we show that:

- If the OLP is available, it should be used, since it solves the RL exploration problem and makes training much faster.
- When OLP is available, RL algorithms may reach the same level of accuracy as OLP methods without using this rich supervised information. They typically require one or two order of magnitude more training iterations on sequence labeling problems.
- When the OLP is not available, RL methods can still be applied. This allows RL methods to solve SP problems – particularly large-scale and complex problem – that cannot be solved by other SP techniques.

Note that, by using approximated RL algorithms, we lose some nice theoretical convergence properties and learning bounds of methods like LaSO and Searn. In approximated value-based RL, we have no convergence proof<sup>5</sup>

---

<sup>5</sup> Except in some special cases that are unrealistic *w.r.t.* our applications [21].

and the algorithms can diverge in some cases. However, in practice, when the feature space is rich enough, divergence does not occur. The OLPOMDP algorithm is known to converge to a local optimum of the expected average reward-per-step. This might be a problem since we have no guaranty to find the globally optimal policy. However, this argument has to be moderated in practice, since we reach state-of-the-art performances on most of our experiments with OLPOMDP.

## 5 Experiments on Sequence labeling

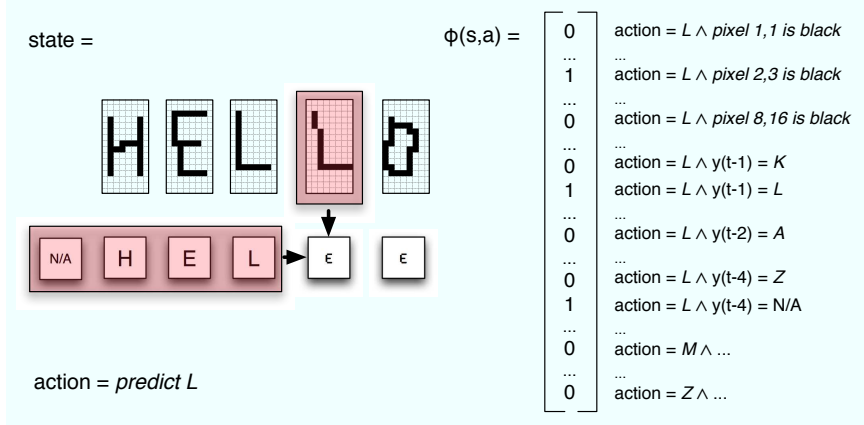
We present below experiments on sequence labeling problems. Sequence labeling is the traditional benchmark for structured prediction methods. The aim of this section is twofold. Firstly, it illustrates the different notions introduced for the description of our approach on a concrete task. Secondly, it allows performing an experimental comparison of several approaches for SP: global models versus incremental models, and state-of-the-art incremental models versus RL incremental models. The results presented in this section extend those of the preliminary work [17].

Let  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_T)$  and  $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_T)$  respectively denote an input sequence and the corresponding label sequence. Label  $\mathbf{y}_t$  corresponds to the observation  $\mathbf{x}_t$  and belongs to the set of possible labels  $\mathcal{L}$ .

### 5.1 SP-MDP Instantiation

We will now detail the initial outputs, the actions and the loss function, in order to fully define the sequence labeling SP-MDP. We introduce a special label  $\epsilon$  for denoting variables  $\mathbf{y}_t$  not yet labeled.

- *Initial Outputs.* The initial partial outputs in sequence labeling are sequences where no label has been decided yet. The initial output  $\bar{\mathbf{y}}_\epsilon$  is thus  $\bar{\mathbf{y}}_\epsilon = (\epsilon, \epsilon, \dots, \epsilon)$ .
- *Actions.* Actions correspond to single label predictions. We have compared two sets of actions corresponding to *left-to-right* labeling and *order-free* labeling. In the former, the first label  $y_1$  is decided at the first step, the second label  $y_2$  at the second step and so forth. At each step, there are  $\text{card}(\mathcal{L})$  possible actions corresponding to all possible labels for the current element. In order-free labeling, any unlabeled element can be labeled at any time (as in Figure 2). This allows the system to first perform the easier decisions (*e.g.* recognize easy letters) in order to gain more contextual information for harder decisions (*e.g.* recognize partially hidden letters).
- *Loss function.* In order to evaluate the quality of a particular labeling, we use the Hamming Loss  $\Delta$ . This loss function simply returns the number of labeling errors:  $\Delta(\hat{\mathbf{y}}, \mathbf{y}) = \text{card}(\{i \in [1, T], \hat{\mathbf{y}}_i \neq \mathbf{y}_i\})$ . In most experiments,



**Fig. 3** Feature function for (left-to-right) sequence labeling actions. The current state contains the input sequence  $\mathbf{x}$  (handwritten letters) and the current prediction  $\bar{\mathbf{y}} = (H, E, L, \epsilon, \epsilon)$ . The current action consists in recognizing the next letter as a  $L$ . The joint description  $\phi(s, a)$  of this state-action pair is given on the right. It contains *content features* (those related to the handwritten digits) and *structural features* (those related to previous predictions).

we use the per-decision reward that reduces to:

$$r_{decision}(\mathbf{s} = (\mathbf{x}, \bar{\mathbf{y}}), \mathbf{a}) = \begin{cases} 1 & \text{if } \mathbf{a} \text{ is a correct labeling action } w.r.t. \mathbf{y} \\ 0 & \text{otherwise} \end{cases}$$

## 5.2 Feature Descriptions

We describe here the feature function  $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$  used for sequence labeling actions. As illustrated in Figure 3, we use content and structural features that lead to large sparse descriptions (*e.g.* from  $10^3$  to  $10^6$  dimensions).

- *Content features.* The content features describe a joint aspect of the action and of the current input element  $\mathbf{x}_t$ . Features related to input elements depend on the task: they may correspond to pixel values in handwritten recognition or word prefixes and suffixes in part-of-speech tagging. In Figure 3, input elements are black-and-white bitmaps and we use one feature  $f_{l,p}$  per possible label  $l \in \mathcal{L}$  and per possible pixel position  $p$ :

$$f_{l,p}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if action label } = l \wedge \text{pixel } p \text{ is black in } \mathbf{x}_t \\ 0 & \text{otherwise} \end{cases}$$

- *Structural features.* The structural features describe the context of the currently predicted label. The context of the label  $\bar{\mathbf{y}}_t$  is  $\{\bar{\mathbf{y}}_{t-C}, \dots, \bar{\mathbf{y}}_{t-1}\}$  in left-right labeling and  $\{\bar{\mathbf{y}}_{t-\frac{C}{2}}, \dots, \bar{\mathbf{y}}_{t+\frac{C}{2}}\}$  in order-free labeling, where  $C$

is a context size parameter. Choosing  $C = 1$  corresponds to the Markov assumption. Larger  $C$  values lead to features describing long-term dependencies. For a label pair  $(l_1, l_2)$ , we use one feature per possible value of  $(l_1, l_2) \in \mathcal{L} \times (\mathcal{L} \cup \{\epsilon\})$  and per possible context position<sup>6</sup>  $\delta$ :

$$f_{l_1, l_2, \delta}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if action label} = l_1 \wedge \bar{\mathbf{y}}_{t-\delta} = l_2 \\ 0 & \text{otherwise} \end{cases}$$

If we consider  $8 \times 16$  black-and-white pixels, 26 possible labels and a context size of 10, there are  $8 \times 16 \times 26 = 3328$  distinct content features and  $26 \times (26 + 1) \times 10 = 7020$  distinct structural features<sup>7</sup>. However, for a given state-action pair, all the features corresponding to labels other than the selected one and those corresponding to white pixels are null. This leads to a sparse representation with, most of the time, less than 100 non-zero features. In practice, sparsity can be exploited efficiently by an implementation that only takes the non-zero features into account. Such sparse high-dimensional representations are commonly used for maximum entropy models [2] or related techniques like CRFs. They are fast to compute and allow us to efficiently use linear learning machines.

### 5.3 Datasets

We performed our experiments on three classical sequence labeling datasets:

- *Spanish Named Entity Recognition (NER)* This dataset, introduced in the CoNLL 2002 shared task<sup>8</sup>, is made of spanish sentences where the aim is to find persons, locations and organisms names (there are 9 distinct labels in  $\mathcal{L}$ ). We used two train/test splits: NER-LARGE (8,324 training sentences) and NER-SMALL (300 training sentences), as in [6] and [24]. Input features include the words, the prefixes and suffixes of the words in a window of +/- 2 words. Statistics for all datasets are provided in Table 1.
- *CHUNK*. This dataset comes from the CoNLL-2000 shared task<sup>9</sup>. The aim is to divide sentences into non-overlapping phrases. In this task, each *chunk* consists of a noun phrase. This task can be seen as a sequence-labeling task thanks to the "BIO encoding": each word can be the Beginning of a new chunk, Inside a chunk or Outside a chunk. This standard dataset put forward by [19] consists of Sections 15-18 of the Wall Street Journal corpus as training material and Section 20 of that corpus as test material. Input

<sup>6</sup> Note that  $\bar{\mathbf{y}}_{t-\delta}$  may sometimes not be defined due to border effects. In order to handle all cases in a uniform way, we introduce a special label, N/A, to denote elements that are beyond the bounds of the sequence.

<sup>7</sup> These are the settings of the HandWritten experiments described in the next part.

<sup>8</sup> <http://www.cnts.ua.ac.be/conll2002/ner/>

<sup>9</sup> <http://www.cnts.ua.ac.be/conll2000/chunking/>

	Training Set		Evaluation Set		Labels	Features
	Sequences	Elements	Sequences	Elements		
NER-SMALL	300	7059	8,323	264,715	9	175,531
NER-LARGE	8,323	264,715	1,517	51,533	9	188,248
HANDWRITTEN-SMALL	626	4,617	6,251	47,535	26	128
HANDWRITTEN-LARGE	6,251	47,535	626	4,617	26	128
CHUNK	8,936	211,727	2,012	47,377	3	143,310

**Table 1** Statistics of the Sequence Labeling corpora. From left to right: number of sequences and elements for the training and testing sets, number of labels and number of input features.

features are similar to those used for the NER dataset plus one additional feature per word that corresponds to the *part-of-speech* of the word.

- *Handwriting Recognition.* This corpus was created for handwriting recognition and was introduced by [15]. It includes 6,600 sequences of handwritten characters that correspond to 6,600 words collected from 150 subjects. Each word is composed of letters, which are  $8 \times 16$  pixels images, rasterized into a binary representation. As in [6], we used two variants of the set: HANDWRITTEN-SMALL is a random split of 10% words for training and 90% for testing. HANDWRITTEN-LARGE is composed of 90% training words and 10% testing words. Letters are described using one feature per pixel, as in Figure 3.

We empirically determined the best values for the context size parameter  $C$ . In the following, we use  $C = 2$  for NER,  $C = 10$  for HANDWRITTEN and  $C = 6$  for CHUNK.

#### 5.4 Baselines

For comparing the approaches, we have performed tests with both independent classification and global SP methods. The former treat each element as an independent prediction problem, while the latter attempt to capture interdependencies between the neighboring elements.

- *Independent Classification.* We give results for maximum entropy classifiers and support vector machines. L1-MAXENT and L2-MAXENT<sup>10</sup> are maximum entropy classifiers, which differ in the regularizer and learner they use. For SVMs, we have used the LIBSVM<sup>11</sup> implementation with linear kernels.

- *Structured Prediction.* We have performed test with two of the global SP methods described in Section 2: CRFs and SVM<sup>struct</sup>. For CRFs, we have

<sup>10</sup> Implementation from <http://nieme.lip6.fr>.

<sup>11</sup> Implementation from <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.

	Independent Classification			SP	
	L2-MAXENT	L1-MAXENT	LIBSVM	CRF	SVM <sup>struct</sup>
NER-SMALL	93.00	92.87	93.26	91.86	<b>93.45</b>
NER-LARGE	96.57	96.75	-	<b>96.96</b>	-
HANDWRITTEN-SMALL	71.65	71.20	<b>77.59</b>	66.86	76.94
HANDWRITTEN-LARGE	78.69	78.56	<b>82.78</b>	75.45	-
CHUNK	96.36	96.56	-	<b>96.71</b>	-

**Table 2** Baselines scores for Sequence Labeling. All the scores are percentages of correctly predicted labels in the test set. The – symbol denotes experiments that failed due to their excessive memory or CPU time requirement.

used the FlexCRFs implementation [18]. For SVM<sup>struct</sup> approach, we have used SVM<sup>hmm</sup><sup>12</sup>.

We tuned each baseline and report the best results on the test set in Table 2.

### 5.5 Experiments

- *Models.* Experiments have been performed with the *left-to-right* and *order-free* models combined with three learning algorithms: SARSA, OLPOMDP (see Section 3.4) and SEARN (see Section 2.3). All methods make use of linear learning machines trained with stochastic ascent/descent. The learning rate is an inversely proportional function of the time. In both SARSA and OLPOMDP, we performed softmax exploration with Gibbs distributions [21]. The temperature of this distribution was chosen to be an inversely proportional function of the time. For each dataset and labeling method, we tuned the following parameters using grid search:

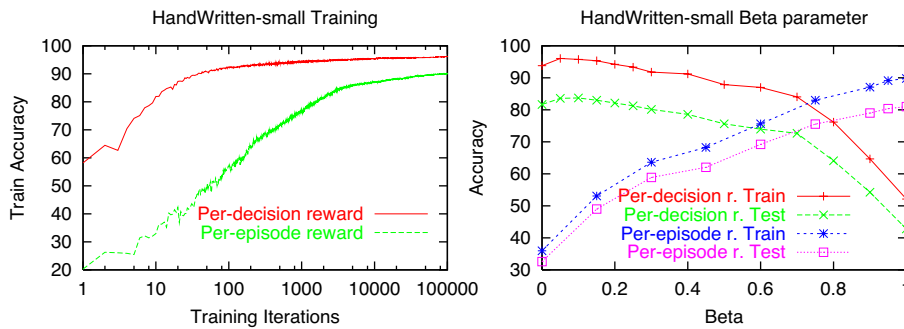
- SARSA: learning rate, temperature, discount
- OLPOMDP: learning rate,  $\beta$  (the bias-variance tradeoff)
- SEARN: learning rate,  $\beta$  (the mixture coefficient between two successive policies)

To evaluate each set of parameters, we performed 100 training iterations<sup>13</sup> with 75% of training data and evaluated the learned policy on the remaining 25% of training data.

- *Per-episode vs. Per-decision Rewards.* In order to evaluate the benefit of decomposing the reward among successive decision steps, we performed a set of experiments comparing per-decision and per-episode rewards. Figure 4 shows the behavior of OLPOMDP depending on which reward function is

<sup>12</sup> Implementation from [http://svmlight.joachims.org/svm\\_struct.html](http://svmlight.joachims.org/svm_struct.html).

<sup>13</sup> In the following, each training iteration corresponds to a whole pass on the training set. The number of RL episodes is the number of training iterations times the training set size.



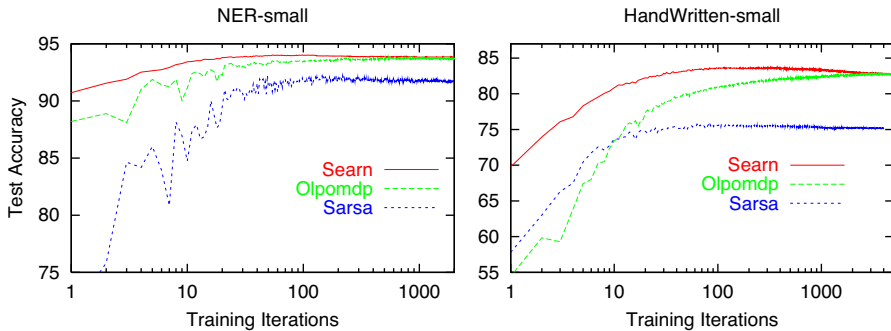
**Fig. 4** Behavior of OLPOMDP depending on which reward function is used. The figures illustrates experiments performed on the HANDWRITTEN-SMALL dataset. Left: train accuracy as a function of the number of training iterations. Right: impact of the  $\beta$  parameter with both rewards on the train and test scores of OLPOMDP.

	Best	Left-Right			Order-Free	
	Baseline	SARSA	OLPOMDP	SEARN	SARSA	OLPOMDP
NER-SMALL	93.45	91.90	93.73	<b>93.83</b>	91.28	93.63
NER-LARGE	96.96	96.31	96.87	<b>97.43</b>	96.32	96.64
HANDWRITTEN-SMALL	77.59	75.20	82.46	83.13	81.56	<b>84.36</b>
HANDWRITTEN-LARGE	82.78	85.88	89.74	90.39	<b>92.21</b>	90.75
CHUNK	96.71	96.08	96.50	<b>96.79</b>	96.17	96.14

**Table 3** Comparison of SARSA, OLPOMDP and SEARN on left-to-right and order-free sequence labeling. Each row corresponds to a dataset and each column corresponds to a learning method. For each combination, we give the percentage of correctly predicted labels on the test-set. The best test scores are shown in bold.

used. Although the per-episode rewards make the learning problem more difficult, OLPOMDP is still able to learn a good policy. However, training requires much more iterations, which makes the per-decision rewards hard to apply in practice. With per-episode rewards, since the whole loss is given at the final states, the best value of the  $\beta$  parameter of OLPOMDP is one, *i.e.* the algorithm requires maximal propagation of the rewards to perform effective learning. At the opposite, when using the per-decision reward, only few propagation of the reward is required, which leads to low optimal  $\beta$  values. In all the remaining experiments, we use the per-decision reward.

- *Global SP vs. Incremental SP.* For each dataset, we performed 1000 iterations of SEARN, SARSA and OLPOMDP and computed the accuracies of the resulting policies on the test set. The results are given in Table 3. On all datasets, the incremental SP methods are competitive with state-of-the-art sequence labeling methods. On the HANDWRITTEN datasets, they even clearly outperform them (thanks to the use of long-term dependencies).



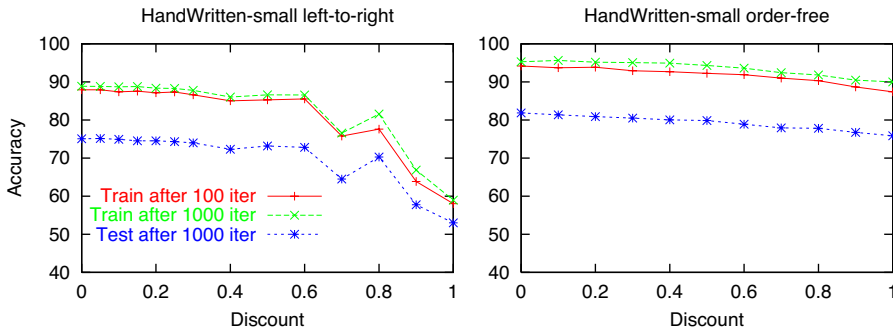
**Fig. 5** Training behavior of SARSA, OLPOMDP and SEARN on the NER-SMALL (left) and HANDWRITTEN-SMALL (right) corpora. For each method, we display the number of correctly predicted labels on the test-set as a function of the number of passes over the training set.

Note that inference in incremental models is several orders faster than with global ones, since it simply consists in greedily executing the learned policy.

- *left-to-right vs. order-free.* Our experiments with order-free labeling actions exhibit two different behaviors. On NER and CHUNK, the increased complexity of order-free labeling slightly degrades the test (and train) accuracies. However, on the handwritten recognition task, order-free labeling significantly improves the prediction accuracies over left-to-right labeling: +4.4% for HANDWRITTEN-SMALL and +3.7% for HANDWRITTEN-LARGE.

- *RL vs. Supervised.* The main difference between SEARN and SARSA or OLPOMDP is the supervision information that is used during training. SEARN relies on the OLP which directly gives local supervision. The RL methods use a weaker supervision in the form of scalar rewards. Our results show that, although using much weaker supervision, the RL methods reach nearly the same accuracy as the supervised SEARN method on most datasets. Figure 5 gives the training curves for each method with left-to-right labeling. On nearly all datasets, we observed the following behavior: SARSA converges towards a sub-optimal solution and OLPOMDP converges to nearly the same result as SEARN, but requires one or two orders of magnitude more training iterations. In summary, removing the rich supervision required by SEARN impacts the required training time but not the accuracy of the model.

- *SARSA Discount.* In order to maximize the total reward criterion, SARSA should be used with a discount factor of 1. We also performed experiments with smaller discount values since this sometimes lead to an increased performance. Figure 6 shows the behavior of SARSA as a function of the discount value on the HANDWRITTEN dataset. In practice, the best discount values were close to zero for most corpora. Since maximizing the immediate per-decision reward leads to an optimal behavior, a null discount seems



**Fig. 6** Impact of the discount parameter in SARSA with left-to-right labeling and order-free labeling. The curves correspond to the training scores after 100 and 1000 training iteration and to the test score after 1000 iterations.

natural for left-to-right labeling. Using a discount factor greater than zero would suggest that some decisions should perhaps go against the correct label for the sake of future reward. In the order-free labeling case, discounting makes more sense: discount factors greater than zero tend to enforce actions that make further predictions easier, *i.e.* between two different correct labeling actions, we should prefer the position that most disambiguate the remaining prediction problems.

- **OLPOMDP  $\beta$  parameter.** The impact of  $\beta$  on OLPOMDP is similar to the one of the discount on SARSA (see Figure 4 for an example). The tuning process led to small, but non-null, values for  $\beta$ : typically from 0.05 to 0.2 for left-to-right labeling and less than 0.15 for order-free labeling.

In this section we illustrated the use of RL to solve sequence labeling tasks. Thanks to greedy inference, all the RL based methods perform very fast inference, while leading to competitive results. The generality of the approach allows us to consider much harder tasks than the simple sequence labeling, such as the transformation task described in the next section.

## 6 Experiments on Tree Transformation

Let us now introduce a new challenging SP task: ordered labeled tree transformation. This task deals with large trees (thousand of nodes), complex transformations (structure and text processing, node creations, deletions and displacements) and very high dimensional learning (often more than one million distinct features). This is a large size and quite complex instance of the more general problem of tree mapping. To our knowledge, there is no method able to deal with this type of problem yet.

This tree transformation task is motivated by the amount of available semi-structured data and the diversity of existing formats for these doc-

uments. Most documents available on the Web are expressed in layout-oriented formats (flat text, wiki-text and HTML), while document-processing applications require more and more semantic-oriented information, in the form of XML dedicated formats for example. This leads to the need for automatic conversion tools between semi-structured document formats [25] with application in domains like document engineering [3] and information retrieval [9, 14]. We propose here to model such conversions as an SP task, where both inputs and outputs are ordered labeled trees. From the point of view of the user, he will simply have to provide a set of documents expressed in both source and target formats, in order to learn the conversion process.

### 6.1 SP-MDP Instantiation

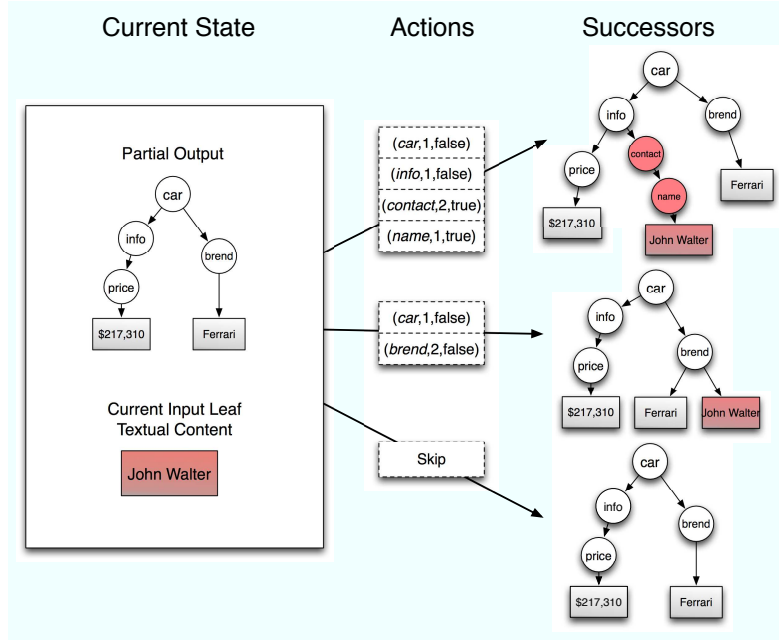
In the following, we model the input tree  $\mathbf{x}$  through its sequence of leaves  $(\mathbf{x}_1, \dots, \mathbf{x}_T)$ . The outputs are rooted labeled ordered trees, whose labels belong to the set  $\mathcal{L}$ .

The key idea of the SP-MDP is that one input leaf  $\mathbf{x}_t$  is processed per time step  $t$ . Several alternative actions may then be selected. Each action of the SP-MDP may involve multiple internal node creations. The input leaf content will be inserted as a leaf of the last created node if any, or as a leaf of an existing output node. Let us now describe the initial outputs, the actions and the loss function that were used in the SP-MDP.

- *Initial Output.* The initial output  $\bar{\mathbf{y}}_\epsilon$  is composed of a single root node. We here assume that all the target trees share the same root label.
- *Actions.* We consider here a generalization of the action set used in [5]. In addition to [5] one allows node displacements and deletions. An action at time  $t$  consists in inserting the textual content  $\mathbf{x}_t$  at some place in the current partial output tree. Formally, we consider that an action  $\mathbf{a}$  is a sequence of triplets  $((l_i, p_i, c_i))_{i \in [1..|a|]}$  where  $l_i \in \mathcal{L}$  corresponds to a possible label of a node,  $p_i$  corresponds to the position of the node *w.r.t.* its parent and  $c_i \in \{true, false\}$  indicates whether the node has to be created or not. Figure 7 illustrates the tree transformation actions. The first example action  $((car, 1, false), (info, 1, false), (contact, 2, true), (name, 1, true))$  corresponds to inserting a subpath  $contact \rightarrow name$  as the second children of the first child  $info$  of the  $car$  root node of the current partial solution. The current input content, here “John Walter”, is inserted as the leaf of the last created node, here  $Info$ . We also consider an additional possible action denoted **Skip** that consists in skipping the current leaf.

The possible action set induced by our formalism is very large (i.e possibly infinite) and we use additional constraints in order to reduce its size:

- We only consider actions that are consistent *w.r.t.* the current partial output tree. For example, using the example in figure 7, the action  $((car, 1, false), (brend, 3, false), (contact, 1, true))$  is not valid *w.r.t.* the



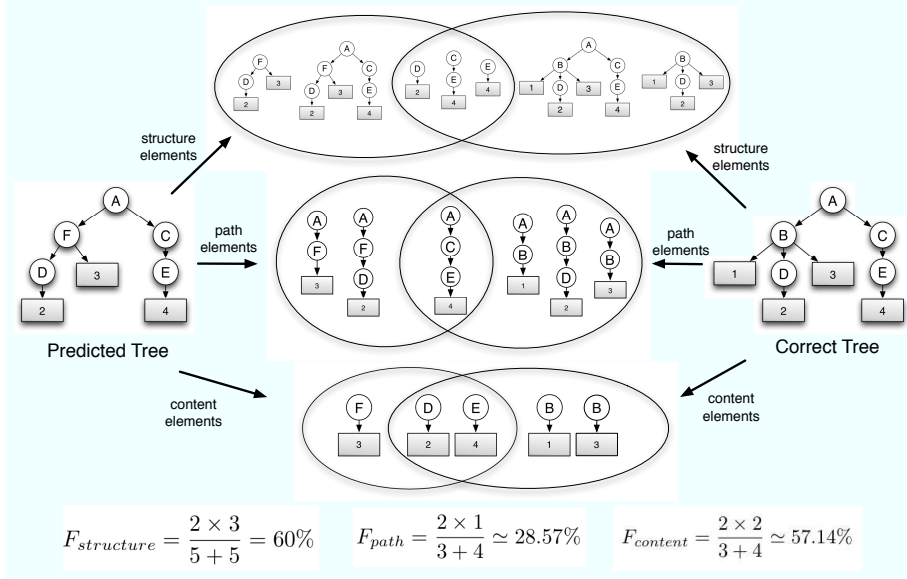
**Fig. 7** Tree Transformation states and actions. The left part of the figure illustrates the current partial output at time  $t = 4$  and the current input leaf  $\mathbf{x}_4 = \text{“John Walter”}$  to be inserted. The middle part of the figure illustrates three possible actions in this state. The first action is composed of four triplets that correspond to the insertion of two new internal nodes plus the current content “John Walter” as a leaf. The second action only inserts the current content as a leaf of an existing node while the third action is the **Skip** action. The right part of the figure gives the new partial output trees corresponding to each action. The nodes that were created by these actions are shaded in red.

partial tree in figure 7 because there is no *brend* node at position 3 under the root of the tree.

- We only consider the actions where the sequence of labels  $(l_1, l_2, \dots, l_{|a|})$  appears at least once in the training set.
- At last, we only consider actions that respect the sequential order of nodes *w.r.t.* the training set *i.e.*: a valid action cannot create a node with label  $l$  as a right sibling of a node with label  $l'$  if there is no node with label  $l$  as a right sibling of a node with label  $l'$  in the training set.

Depending of the tree transformation datasets, we also consider additional constraints that reduce the size of the set of possible actions (additional details are provided in the experiments section):

- **PathOccurrencesThreshold(n)** (POT) Only the label paths that appear at least  $n$  times in the training output trees are considered.



**Fig. 8** Computation of the  $F_{structure}$ ,  $F_{path}$  and  $F_{content}$  similarity measures. The left and right parts of the figure respectively correspond to the predicted tree and the correct tree. For each similarity measure, the trees are decomposed into set of elements (structure elements, path elements or content elements). The bottom part of the figure gives the similarity scores, which are the  $F_1$  score between predicted and correct elements.

- **MoveMaxDistance(n)** (MMD) Large displacement of nodes are disabled: when processing the leaf  $x_t$ , only actions that put the text under leaves with indices in  $[t - n, t + n]$  are allowed.
- **NoSkip** (NS) This constraint removes the **Skip** action.

At each step of the process, the set of possible action is enumerated recursively by a simple procedure. Note that, in the proposed experiments, we obtained up to 5,000 distinct actions per state.

• **Loss function.** In order to evaluate the quality of the predicted outputs, we make use of three tree-similarity  $F1$  scores, which are illustrated in Figure 8. The  $F_{structure}$ ,  $F_{path}$  and  $F_{content}$  measures are computed by decomposing each tree into a set of parts and by computing the  $F_1$  scores<sup>14</sup> on these decompositions. The three  $F_1$  scores are always in the interval  $[0, 1]$  and perfectly predicted trees lead to similarity scores of 1.  $F_{content}$  focuses on the proportion of correctly labeled leaves,  $F_{path}$  concerns the proportion of correctly recovered paths and  $F_{structure}$  concerns the proportion of correctly recovered subtrees. The loss function used for learning is:

$$\Delta(\hat{\mathbf{y}}, \mathbf{y}) = 1 - F_{structure}(\hat{\mathbf{y}}, \mathbf{y})$$

<sup>14</sup> The  $F_1$  score between two sets  $a$  and  $b$  is:  $F_1(a, b) = \frac{2 \times \text{card}(a \cap b)}{\text{card}(a) + \text{card}(b)}$ .

### 6.2 Feature Descriptions

In order to describe tree transformation state-action  $\phi(\mathbf{s}, \mathbf{a})$ , we used a large set of sparse binary features. These features are described in details in Appendix and we just briefly give here a general description. Note that these features follow the same idea than the ones for the sequence labeling task (see Section 5.2). The features are created with a few hand defined feature generators and lead to sparse descriptions in a very large feature space. For the tree transformation task, a new feature subset is generated by considering sequentially each triplet defining an action. The feature subsets are then gathered in order to obtain the full feature description. For each triplet  $(l_i, p_i, c_i)$ , we generate four kind of features:

- **Input Content Features** describe the current textual content.
- **Input Path Features** describe the structural context of the current node in the input tree.
- **Prediction Path Features** describe parent labels of the new nodes in the output-tree.
- **Prediction Neighborhood Features** describe sibling labels of the new nodes in the output-tree.

### 6.3 Complexity of the task

Tree transformation is a particularly challenging large-scale SP task with several real-world applications. We provide below a short discussion on why most existing SP algorithms cannot cope with the complexity or the scale of such a task.

• *Global SP models and tree transformation.* Global SP models assume that the *argmax* problem (see Section 2.2 and Section 4) can be solved efficiently at each learning iteration. In tree transformation, the *argmax* computation is intractable for two reasons:

- The loss function  $F_{structure}$  is not additively decomposable. Solving the *argmax* thus leads to a complex optimization problem.
- Even with a simpler loss and a simpler transformation task, the optimization is intractable with dynamic programming algorithms. For example, [25] and [3] propose methods using dynamic programming on a restricted tree transformation problem (one-to-one tree mapping: leaf nodes can nor be displaced, neither suppressed). Due to their cubic complexity in the number of leaves, those dynamic programming methods do not scale to document containing thousands of leaves such as ours.

• *Incremental SP models and tree transformation.* As discussed in Section 4, other incremental SP models make assumptions like OLT or OLP availability. In some cases, when the output space is complex, computing OLP - i.e. *given any partial output (which may contain errors) find the best action to take* is not a trivial problem. In such cases, the authors of SEARN suggest

Corpus	Formats	Size	Internal Nodes	Leaves	Depth	Labels
REALESTATE	XML → XML	2,367	≈ 33	≈ 19	≈ 6	37
MIXED-MOVIE	HTML → XML	13,048	≈ 64	≈ 39	5	35
SHAKESPEARE	Text → XML	750	≈ 236	≈ 194	≈ 4.3	7
INEX-IEEE	Text → XML	12,107	≈ 650	≈ 670	≈ 9.1	139
WIKIPEDIA	HTML → XML	10,681	≈ 200	≈ 160	≈ 7.7	256

**Table 4** Tree transformation corpora statistics. From left to right: the name of the corpus, the input and output formats, the number of documents, the mean number of internal nodes per tree, the mean number of leaves per tree, the mean tree depth and the number  $card(\mathcal{L})$  of output labels.

to use a search-procedure to (approximately) find the best action. Consider for example a greedy beam-search procedure. The complexity of searching the optimal actions for one training trajectory<sup>15</sup> is  $\mathcal{O}(T^2 \times b \times a^2)$ , where  $b$  is the size of the beam,  $T$  is the depth of the search (in our case the number of input leaves) and  $a$  is the mean number of available actions per state.

On large-scale tasks such as tree transformation, determining an approximate OLP with beam search has a prohibitive complexity. Let us perform a naive computation of the time required to process one typical document with  $T = 100$  leaves and up to  $a = 5,000$  actions per state with a beam size of 5. On such documents, our implementation performs inference – with  $\mathcal{O}(T \times a)$  complexity – in  $\approx 1s$ . The time required to search the best action in each visited state for one document is thus of the order of  $1s \times 100 \times 5000 \times 5 \approx 1$  month computation time. This solution is then clearly intractable. On the other hand, the RL formalism provides algorithms that can naturally handle such complex SP problems because they do not rely on any OLP computation.

#### 6.4 Datasets

We used three medium-scale datasets and two large-scale datasets whose statistics are summarized in Table 4.

- REALESTATE [10]. This corpus, proposed by Anhai Doan<sup>16</sup> is made of 2,367 data-oriented XML documents. The documents are expressed in two different XML formats. The aim is to learn the transformation from one format to the other.
- MIXED-MOVIE [9]. The second corpus is made of more than 13,000 movie descriptions available in three versions: two mixed different XHTML versions and one XML version. This corresponds to a scenario where two dif-

<sup>15</sup> Given a state-action pair, finding the best completion of the partial output, takes  $\mathcal{O}(T \times b \times a)$  time. This search has to be launched for each action  $a \in \mathcal{A}_s$  in each visited state  $s$  of the trajectory:  $\mathcal{O}(T \times a)$  times.

<sup>16</sup> <http://www.cs.wisc.edu/~anhai/>

Corpus	Constraints			Features			
	MMD	NS	POT	Content	In. Path	Pred. Path	Pred. Neigh.
REALESTATE	$\infty$		0	0	2	2	2
MIXED-MOVIE	10		0	6	6	6	4
SHAKESPEARE	0	✓	0	6	0	5	6
INEX-IEEE	0	✓	0	6	0	5	6
WIKIPEDIA	10		5	6	6	6	4

**Table 5** Tree transformation SP-MDP constraints and feature parameters. From left to right: the dataset name, the MoveMaxDistance constraint, the NoSkip constraint, the PathOccurrencesThreshold constraint, the context size for content features, the context size for input path features, the context size for prediction path features and the context size for prediction neighborhood features. All parameters here have been roughly tuned on the training sets.

ferent websites have to be mapped onto a predefined mediated schema. The transformation includes node suppression and some node displacements.

- SHAKESPEARE [3]. This corpus is composed of 60 Shakespeare scenes<sup>17</sup>. These scenes are small trees, with an average length of 85 leaf nodes and 20 internal nodes over 7 distinct tags. The documents are given in two versions: a flat segmented version and the XML version. The tree transformation task aims at recovering the XML structure using only the text segments as input.
- INEX-IEEE [11]. The INEX-IEEE corpus is composed of 12,017 scientific articles in XML format, coming from 18 different journals. The tree transformation task aims at recovering the XML structure using only the text segments as input.
- WIKIPEDIA [8]. This corpus is composed of 12,000 wikipedia pages. For each page, we have one XML representation dedicated to wiki-text and the HTML code. The aim is to use the layout information available in the HTML version, for predicting the semantical XML representation.

For each corpus, we randomly extracted 100 examples to form the training set. As mentioned in Section 6.1, we use some constraints to limit the number of possible actions. These constraints are given in Table 5. We also give in this table the context size parameters that were used into the feature function  $\phi$ . Few tuning effort was spent on these parameters. For the REALESTATE dataset, the transformation that has to be learned only depends on the labels, so we removed the content features. For the SHAKESPEARE and INEX-IEEE datasets, the input documents are flat segmented documents, so that input path features are unavailable.

### 6.5 Experiments

- *Models.* We applied the SARSA and OLPOMDP algorithms to our tree transformation corpora by using the same settings as for sequences. We also

<sup>17</sup> <http://metalab.unc.edu/bosak/xml/eg/shaks200.zip>

used the same tuning process as previously, with 100 training iterations for each trial.

- *Baselines.* We only have one *learning-based* baseline on two of these datasets for the complexity reasons explained in Section 6.3. The baseline model PCFG+ME [3] is a global model for the *restricted* tree transformation problem where the order of the leaves in the input tree and the corresponding output tree are exactly the same. It models the probability of outputs by using probabilistic context free grammars (PCFG) and maximum-entropy (ME) classifiers. Inference is then performed with a dynamic-programming algorithm that has a cubic complexity in the number of input leaves. This model can only operate on the two smallest datasets REALESTATE and SHAKESPEARE.

In order to compare our scores on all datasets, we performed a set of experiments by running non-learning baselines using a random policy  $\pi^{random}$  – which selects actions randomly – and two greedy policies  $\pi_{structure}^{greedy}$  and  $\pi_{path}^{greedy}$ . These greedy policies make use of the correct output and select actions whose execution most increase the  $F_{structure}$  or  $F_{path}$  similarity scores. For example,  $\pi_{structure}^{greedy}$  is defined in the following way:<sup>18</sup>

$$\pi_{structure}^{greedy}(\mathbf{s} = (\mathbf{x}, \bar{\mathbf{y}})) = \underset{\mathbf{a} \in \mathcal{A}_s}{\operatorname{argmax}} F_{structure}(\mathbf{a}(\bar{\mathbf{y}}), \mathbf{y})$$

Note that, since they rely on the correct outputs, the baseline greedy policies cannot generalize on new examples. The three scores of  $\pi_{structure}^{greedy}$  could be considered as upper-bounds for the performance reachable by learning to maximize the immediate reward, *e.g.* with SARSA and a discount factor of 0.

- *Medium-scale results.* Our experimental results on the three medium-scale datasets are given in Table 6. On REALESTATE and MIXED-MOVIE, the RL approaches give significantly better results than the greedy policy baselines. This result is very satisfactory and has two major implications. Firstly, it shows that RL algorithms are able to find a *better strategy than greedily moving* toward the correct output. Secondly, it shows that the algorithms perform an *effective learning and generalization* of this strategy. On SHAKESPEARE, the scores of RL algorithms is slightly inferior to those of the greedy policies. Since greedy policies perform nearly perfectly on this corpus, the main difficulty here is more related to *generalization* than *exploration*.

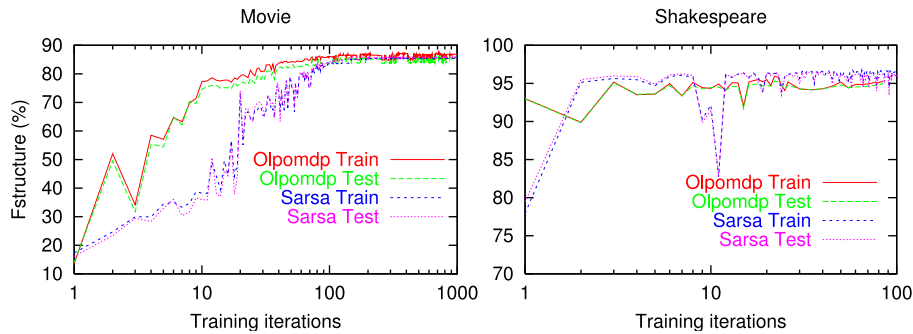
- *Comparison with state-of-the-art.* The REALESTATE collection corresponds to an *XML database* where we need to label the leaves correctly and to put them in the correct order. The task is easy, and most RL approaches achieve > 99% on the different scores. PCFG+ME only performs 7 % on the  $F_{path}$  score and about 50 % on the  $F_{structure}$  score for two reasons: this model is based on the MMD(0) constraint (see part 6.1) which

---

<sup>18</sup> If multiple actions equally increase the similarity scores, one of these actions is selected randomly.

Corpus	Score	RL		Baselines			PCFG+ME
		SARSA	OLPOMDP	$\pi_{structure}^{greedy}$	$\pi_{path}^{greedy}$	$\pi^{random}$	
REALESTATE	$F_{structure}$	99.54	<b>99.99</b>	87.09	<b>97.09</b>	3.27	49.8
	$F_{path}$	99.87	<b>99.99</b>	84.42	<b>100</b>	3.91	7
	$F_{content}$	99.88	<b>100</b>	<b>100</b>	<b>100</b>	5.10	99.9
MIXED-MOVIE	$F_{structure}$	86.22	<b>86.50</b>	<b>47.04</b>	44.15	3.54	/
	$F_{path}$	91.53	<b>91.88</b>	52.02	<b>52.18</b>	5.29	/
	$F_{content}$	91.53	<b>92.05</b>	52.02	<b>52.18</b>	5.67	/
SHAKESPEARE	$F_{structure}$	<b>96.03</b>	95.88	<b>98.65</b>	75.16	11.34	94.7
	$F_{path}$	<b>97.88</b>	97.72	98.91	<b>100</b>	16.47	97.0
	$F_{content}$	<b>98.87</b>	98.40	99.83	<b>100</b>	18.25	98.7

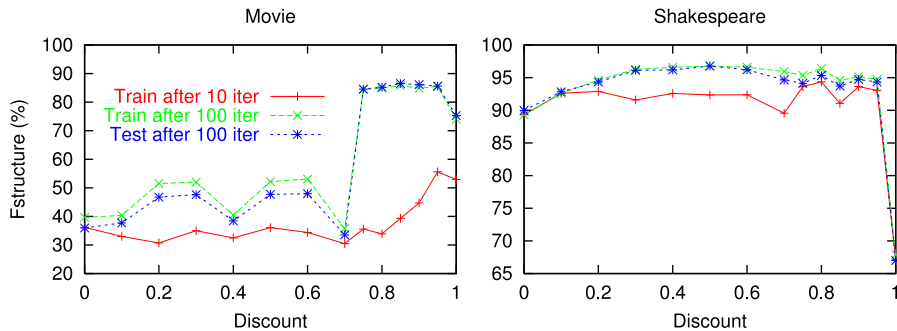
**Table 6** Tree transformation results for medium-scale datasets. For each dataset and each method, we give the three average similarity scores  $F_{structure}$ ,  $F_{path}$  and  $F_{content}$  between the predicted and correct trees of the test set. The two first columns correspond to the SARSA and OLPOMDP RL algorithms. The next three columns are baselines which do not rely on learning. The last column is the PCFG+ME [3] baseline. The / symbol denotes results that could not be computed due to the complexity of PCFG+ME.



**Fig. 9** Training behavior of SARSA and OLPOMDP on the MIXED-MOVIE (left) and SHAKESPEARE (right) corpora. We give the train and test  $F_{structure}$  scores for each method as a function of the number of training iterations.

is not verified in this collection and PCFG+ME is not able to delete input leaves, which is required here. On the SHAKESPEARE corpus, PCFG+ME gives slightly lower results than RL methods. This result might be related to the different kind of features exploited by the methods. Since PCFG+ME relies on a PCFG, it cannot deal with some of the features that we use here (e.g. the prediction path features, see Appendix).

- *Training behavior.* The training behavior of SARSA and OLPOMDP is illustrated by Figure 9. We have often observed that the training behavior of SARSA is slightly more noisy than that of OLPOMDP. On our datasets, the number of training iterations needed to converge was reasonable in all cases.



**Fig. 10** Impact of the discount parameter in tree transformation with SARSA. The curves correspond to the training scores after 100 and 1000 training iterations and the test scores after 1000 training iterations.

Corpus	Score	RL	Baselines		
		SARSA	$\pi_{structure}^{greedy}$	$\pi_{path}^{greedy}$	$\pi^{random}$
INEX-IEEE	$F_{structure}$	67.5	<b>76.32</b>	49.94	2.17
	$F_{path}$	74.4	39.23	<b>97.20</b>	1.00
	$F_{content}$	75.8	82.91	<b>97.20</b>	8.62
WIKIPEDIA	$F_{structure}$	<b>65.6</b>	57.37	23.53	5.51
	$F_{path}$	<b>74.3</b>	2.28	32.28	0.12
	$F_{content}$	<b>80.2</b>	72.92	39.34	12.35

**Table 7** Large scale tree transformation. For each method and each dataset, we give the three average similarity scores on the test set. We compare SARSA with the baseline policies, which do not use learning.

- *SARSA Discount.* The impact of the discount factor on SARSA is illustrated in Figure 10. The optimal discount factor values highly depend on the corpus. On some corpora, such as MIXED-MOVIE, discount factors close to 1 are required to outperform the greedy policy baselines. In some other cases, such as SHAKESPEARE, the optimal discount factors are close to 0.5. As for sequences, the optimal discount factors are not 1. We believe that discount factors smaller than 1 lead to simpler exploration problems, which makes learning easier. This may also be related to the use of SARSA with a limited amount of training material. The study of the number of training examples on the behavior of RL algorithms is left for future work.

- *OLPOMDP  $\beta$  parameter.* As for sequences,  $\beta$  has a similar impact on OLPOMDP as the discount on SARSA has. For the MIXED-MOVIE and SHAKESPEARE corpora, the best  $\beta$  value were respectively 0.85 and 0.5.

- *Large-scale tree transformation.* In order to demonstrate the scalability of the RL approach, we have performed experiments with SARSA on the two large-scale corpora. Since, in our implementation, episodes with

OLPOMDP take about 50 more time than episodes with SARSA<sup>19</sup>, we did not use OLPOMDP on the large-scale datasets. These experiments required  $\approx 5$  days training time in order to perform 1000 training iterations, *i.e.*  $10^5$  RL episodes, for each dataset using SARSA. This huge amount of time has to be contrasted with the scale of the task: these corpora involve particularly large documents (the biggest documents in these corpora contain up to 10.000 nodes), complex operations (nodes displacements or nodes deletions), highly heterogeneous documents and large number of labels (139 labels for INEX-IEEE and 256 labels for WIKIPEDIA). The results are given in Table 7. On WIKIPEDIA, SARSA outperforms the scores of the greedy policy baselines, which is very satisfactory, given the large number of labels on this corpus. On INEX-IEEE, SARSA does not reach the level of the greedy policy baselines. Note that this corpus contains a huge amount noise.

To summarize, these experiments emphasize three interesting characteristics of the RL approach for tree transformation:

- Scalability: the SARSA method is able to learn with all our large-scale real-world datasets.
- Inference speed: most documents are processed in less than one second, which, on the small datasets, is about 50 times faster than PCFG+ME inference.
- Relevancy of RL: on most datasets, RL algorithms find better strategies than the greedy behavior and succeed in learning and generalizing these strategies.

## 7 Conclusion

We have proposed a formalization of the supervised structured prediction problem as a Markov Decision Process. We have shown that solving the policy for this Decision Process is equivalent to optimizing the empirical loss for supervised learning. Thanks to this relation, one can use approximate RL algorithm for solving SP. This provides a principled approach for a large range of SP problems. The proposed model is general and makes fewer assumptions than most existing SP models. It can be used for solving complex and large-scale SP problems. The performance of our approach was demonstrated on two kinds of SP problems: sequence labeling and tree transformation. From the point of view of RL, the proposed MDP has some unique and unusual features. This example also shows that RL can be used to handle large-scale problems expressed in very high dimensional spaces. This work opens the way for exploring the potential of RL methods and their many variants for SP.

---

<sup>19</sup> SARSA runs much faster thanks to a particularity of our implementation, which is able to compute  $\langle \phi(\mathbf{s}, \mathbf{a}), \theta \rangle$  values without storing the  $\phi(\mathbf{s}, \mathbf{a})$  vectors in memory. In OLPOMDP, the main cpu-time bottleneck is the allocation and deletion of data structures for storing these vectors.

## References

1. Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:2001, 2001.
2. Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. A maximum entropy approach to natural language processing. In *Computational Linguistics*, 1996.
3. Boris Chidlovskii and Jérôme Fuselier. A probabilistic learning method for xml annotation of documents. In *IJCAI*, 2005.
4. Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.
5. Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July 2004.
6. Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Submitted to the Machine Learning Journal*, 2006.
7. Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005. ACM Press.
8. Ludovic Denoyer and Patrick Gallinari. The wikipedia xml corpus. *SIGIR Forum*, 2006.
9. Ludovic Denoyer and Patrick Gallinari. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. *SIGIR Forum*, 41(1):79–90, 2007.
10. Anhai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Maching Learning*, 50(3):279–301, 2003.
11. Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors. *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX), Schloss Dagstuhl, Germany, December 9-11, 2002*, 2002.
12. Frédéric Garcia and Seydina M. Ndiaye. A learning rate analysis of reinforcement learning algorithms in finite-horizon. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 215–223, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
13. Amir Globerson, Terry Koo, Xavier Carreras, and Michael Collins. Exponentiated gradient algorithms for log-linear structured prediction. In *ICML*, pages 305–312, 2007.
14. Florent Jousse, Rmi Gilleron, Isabelle Tellier, and Marc Tommasi. Conditional random fields for xml trees. In *ECML Workshop on Mining and Learning in Graphs*, 2006.
15. Robert Howard Kassel. *A comparison of approaches to on-line handwritten character recognition*. PhD thesis, Cambridge, MA, USA, 1995.
16. John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.
17. Francis Maes, Ludovic Denoyer, and Patrick Gallinari. Sequence labelling with reinforcement learning and ranking algorithms. In *ECML*, Warsaw, Poland, 2007.

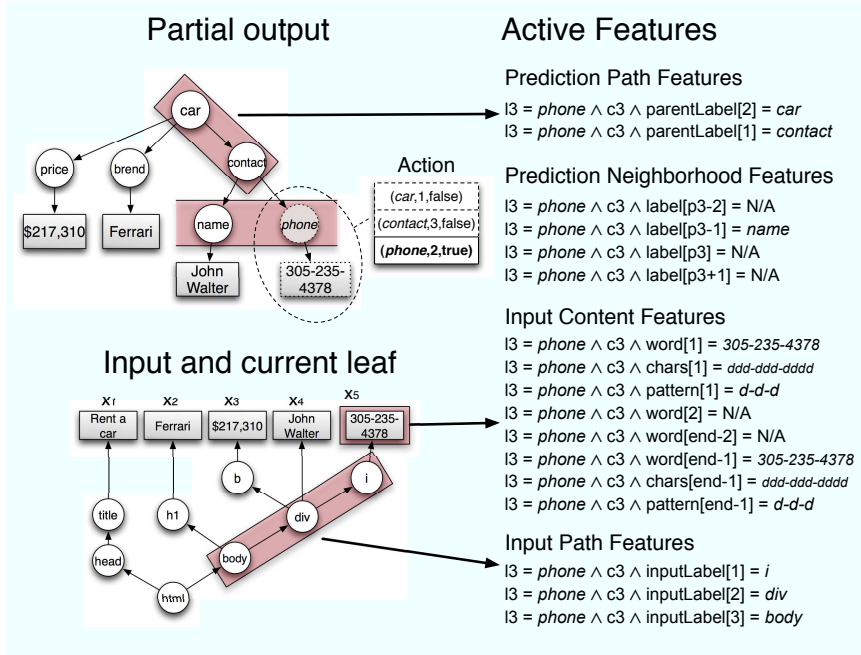
18. Xuan-Hieu Phan and Le-Minh Nguyen. Flexcrfs: Flexible conditional random field toolkit, 2005. <http://flexcrfs.sourceforge.net>.
19. Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey, 1995. ACL.
20. Walter L. Ruzzo. On the complexity of general context-free language parsing and recognition. In *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*, pages 489–497, London, UK, 1979. Springer-Verlag.
21. R. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. MIT Press, 1998.
22. Benjamin Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *NIPS*, 2003.
23. Ivan Titov and James Henderson. Incremental bayesian networks for structure prediction. In *ICML*, pages 887–894, 2007.
24. Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, New York, NY, USA, 2004. ACM Press.
25. Guillaume Wisniewski, Ludovic Denoyer, Maes Francis, and Patrick Gallinari. Probabilistic model for structured document mapping. In *5th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM'07)*, Germany, 2007.

## A Appendix

The features correspond to the joint representation of both an action  $\mathbf{a}$  and a state  $\mathbf{s}$ . For tree transformation, actions are sequences of triplets  $(l_i, p_i, c_i)$ . For each triplet  $(l_i, p_i, c_i)$ , we generate a set of features describing jointly the current triplet and the state  $\mathbf{s}$ . All the triplet features are then concatenated in order to obtain the action features. These features are mapped onto the vector representation  $\phi(\mathbf{s}, \mathbf{a})$  of the  $\mathbf{s}, \mathbf{a}$  pair. Figure 11 illustrates the features corresponding to the third triplet  $(phone, 2, true)$  for action  $\mathbf{a} = ((car, 1, false), (contact, 3, false), (phone, 2, true))$ . There are four kinds of features used in the experiments:

*Input Content Features* describe the textual content of the current input leaf. The input leaf content is first summarized by the  $p$  first and the  $p$  last leaf word where  $p$  is a context parameter which depends on the application. Then, features are computed only for these words. There are three types of features called *word*, *character type* and *pattern*:

- **Word features** are computed for each word: there is one feature denoted  $f_{i,c,l,j,w}$  for each triplet  $i \in [1, |\mathbf{a}|]$ , output label  $l \in \mathcal{L}$ , creation flag  $c \in \{true, false\}$ , word position  $j$  in the leaf word sequence and



**Fig. 11** Tree transformation action features. The top-left part of the figure gives the current partial output and a candidate action. The candidate action consists in creating a *phone* node into the *contact* node. The bottom-left part of the figure gives the input tree and the current input leaf  $x_5$ . The right part of the figure lists the active features corresponding to the third triplet of the action:  $(\text{phone}, 2, \text{true})$ . The active features are those which have a value of 1. The rectangles in red indicate the context considered for each feature type. This context is the number of ancestors, siblings or adjacent words considered for computing the features. Since there is only one word in the current textual content, the same word appears in the *first words* features and the *last words* features.

word  $w$ :

$$f_{i,c,l,j,w}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } l_i = l \wedge c_i = c \wedge \text{inputWord}[j] = w \\ 0 & \text{otherwise} \end{cases}$$

where  $\text{inputWord}[j]$  is the word number at position  $j$  in the current leaf.

- **Character type features** correspond to the type of characters used in words. The type of a character can be a digit, an upper case or lower case letter of the alphabet, or other. The character type features of a word are constructed by replacing each lower letter of the word by 'l', each upper letter by 'u', each digit by 'd' and other by 'o'.
- **Pattern features** of a word are built based on the **Character type features** by replacing successive occurrences of the same character type by a unique occurrence. This roughly corresponds to a regular expres-

sion over the types of characters of a particular word. Figure 11 gives examples of such features.

*Input Path Features* Input Path Features correspond to the structural context of the current input leaf  $x_i$ . They encode the leaf path in the input tree. There is one such feature per triplet index  $i \in [1, |\mathbf{a}|]$ , creation flag  $c \in \{true, false\}$ , output label  $l \in \mathcal{L}$ , height  $h$  in the input tree and possible label  $l'$  in the input tree.

$$f_{i,c,l,h,l'}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } l_i = l \wedge c_i = c \wedge \text{inputLabel}[h] = l' \\ 0 & \text{otherwise} \end{cases}$$

where  $\text{inputLabel}[h]$  is the label of the ancestor of the current leaf at height  $h$ . The last two types of features encode node information in the partial output tree.

*Prediction Path Features* Prediction Path Features encode the description of the node corresponding to the triplet  $(l_i, p_i, c_i)$  and of its ancestors. There is one such feature per triplet index  $i \in [1, |\mathbf{a}|]$ , creation flag  $c \in \{true, false\}$ , pair of labels  $(l, l') \in \mathcal{L}^2$  and ancestor height  $h$ :

$$f_{i,c,l,l',h}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } l_i = l \wedge c_i = c \wedge \text{parentLabel}[h] = l' \\ 0 & \text{otherwise} \end{cases}$$

where  $\text{parentLabel}[h]$  is the label of the ancestor of the node concerned by triplet  $i$  at height  $h$ .

*Prediction Neighborhood Features* These features are similar to the prediction path features, except that they encode the sibling labels of the node to be inserted instead of its parent labels. For each triplet index  $i \in [1, |\mathbf{a}|]$ , creation flag  $c \in \{true, false\}$ , label pair  $(l, l') \in \mathcal{L}^2$  and context position  $\delta$ , one defines a feature as:

$$f_{i,c,l,l',\delta}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } l_i = l \wedge c_i = c \wedge \text{neighboringLabel}[p_i + \delta] = l' \\ 0 & \text{otherwise} \end{cases}$$

where  $\text{neighboringLabel}[p_i + \delta]$  corresponds to the label of the sibling of node concerned by the triplet  $i$  at position  $p_i + \delta$ .

Not all the siblings and ancestors are considered while computing the features. The number of sibling and ancestors considered in the experiments is provided in Table 5 and correspond to the context of the node illustrated in Figure 11

At last we introduce a special label called N/A to denote out-of-bounds elements, *i.e.* elements that do not exist in the tree. Typically, if we consider a leaf with only one word and we want to compute the feature of the second

word (see *Input Content Features*) of the leaf which does not exist, we will use this special label. This is illustrated in Figure 11.

Our four kind of features lead to very high dimensional feature spaces: there are often more than  $10^6$  distinct features. However, for a given state-action pair, the number of non-null features is relatively low (*e.g.* less than 100 non-null features). Once again, sparsity can be exploited through a smart implementation considering only non-null features. All the features are generated automatically from the data, using the above definitions.