Generic Oracles for Structured Prediction

Christoph Teichmann
Bloomberg
cteichmann1@bloomberg.net

Antoine Venant
Université Montreal
antoine.venant@umontreal.ca

Abstract

When learned without exploration, local models for structured prediction tasks are subject to exposure bias and cannot be trained without detailed guidance. *Active Imitation Learning* (AIL), also known in NLP as Dynamic Oracle Learning, is a general technique for working around these issues by allowing the exploration of different outputs at training time.

AIL requires *oracle feedback*: an oracle is any algorithm which can, given a partial candidate solution and gold annotation, find the correct (minimum loss) next output to produce.

This paper describes a general finite state technique for deriving oracles. The technique described is also efficient and will greatly expand the tasks for which AIL can be used.

1 Introduction

Structured Prediction tasks, e.g., POS tagging, machine translation or syntactic parsing, are central to NLP and are commonly solved with machine learning based models. There are two main ways of approaching these problems: in one, a model scores fragments of possible outputs and an efficient decoding algorithm finds the highest scoring solution, e.g., using conditional random fields and the forward-backward algorithm. In the second approach a model produces an output through a sequence of decisions, each extending a partial output produced by the previous steps, e.g., picking one word after the other in a sequence to sequence translation system or repeatedly splitting a sentence into constituents. Modern neural models use complex hidden states to express the interdependence between outputs, making efficient decoding difficult and the latter approach ever more important.

When training models to make sequential decisions it is necessary to provide guidance on which actions to take to achieve minimum loss against a gold output. Sometimes there is a clear sequence of correct actions, e.g., when learning to translate or tag, there is the option of simply training the model to follow the gold annotation, which corresponds 1-1 to possible model outputs.

1.1 The Problems Dynamic Oracles Solve

Not all tasks have straightforward gold sequences. Consider the problem of simplifying a sentence by tagging words either to be deleted or replaced with more common, semantically similar words. There may be multiple ways to simplify that generate the same end result. If only a gold simplification is annotated, and no gold sequence of **actions** (*i.e.* deletion or replacement), then it is not clear which sequence of actions to train for. For another example, consider multiple annotations coming from multiple annotators, where it is necessary to interpolate between them.

Furthermore, when only following gold sequences, the model will never learn to recover from incorrect choices, as they are not encountered during training - the so called exposure bias. Consider the following example: assume that we want to map a sentence to a parse tree as in Fig. 1. For a simple sequence to sequence model, a parse tree is produced by outputting opening and closing brackets as well as words and mapping the result to a tree. If the model incorrectly added an NP(bracket right before “hit”, then a gold sequence based training would never expose the model to a similar situation. The model would have no knowledge of how to recover from the error with minimal loss, and how to best represent a sequence with a questionable bracket.

Both exposure bias and the absence of clear gold training sequences can be tackled with *active imitation learning* (AIL). AIL uses a source of ground truth to determine the optimal action to take at each step. These sources of ground truth are called *dy-*

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 1–12
Bangkok, Thailand (online), August 6, 2021.

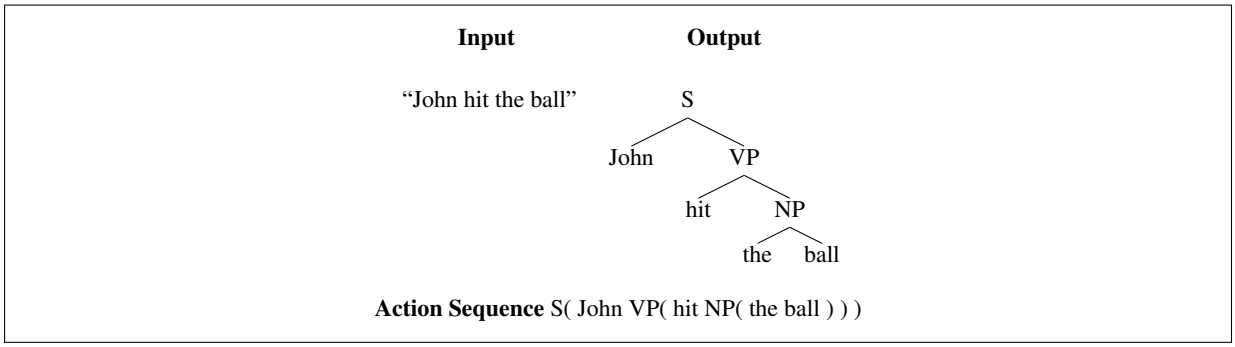


Figure 1: Example structured prediction task for active imitation learning.

namic oracles in the NLP literature, or *experts* in AIL focused works. Dynamic oracles determine what to do for any partially complete solution to minimize the loss relative to a gold annotation and they contrast with static oracles, which only provide a gold output sequence.[1] This paper is concerned with a generic way to build dynamic oracles for NLP problems. In our example of an incorrectly placed NP(, AIL would enable us to create training examples that contain similar errors and show how to recover from them.

1.2 Contribution

Dynamic oracles have been developed for different parsing tasks (Goldberg and Nivre, 2012; Goldberg et al., 2014; Coavoux and Crabbé, 2016; Fernández-González and Gómez-Rodríguez, 2018b; Coavoux and Cohen, 2019; Gómez-Rodríguez and Fernández-González, 2015) and have been shown to improve parsing performance (Ballesteros et al., 2016; Goldberg and Nivre, 2012; Coavoux and Crabbé, 2016; Fernández-González and Gómez-Rodríguez, 2018b). These oracles work for specific output types and losses. It is sometimes possible to use an oracle derived for one problem in a different context, but this transfer is limited. Here we instead give a completely generic technique for deriving dynamic oracles.

We focus on problems that involve mapping an input sequence to an output sequence in left to right order, which also generalizes the task of tagging the sequence. Our approach is general enough to subsume others, e.g., parsers based on transition systems can be encoded through tagging (Gómez-Rodríguez et al., 2020). Our technique is based on encoding possible outputs in a finite state automaton. By incorporating the loss via a transducer, we are able to formulate oracles as a minimum weight problem on regular languages. We also investigate the complexity of repeatedly solving these minimum weight problems.

2 Formal background

Before we describe our approach, we will recap some of the theory of AIL and finite state machines. Through a detailed discussion of both topics in a shared vocabulary, the connection will become clearer. We use the task of mapping sentences to parse trees as our running example.

General Notation We start with generic notations that will be used throughout the paper: we denote by $[k, n]$ the set of natural numbers between k (included) and n (included). For any set Σ we let $\wp(\Sigma)$ denote the powerset (set of all subsets) of Σ, and Σ^* denote the set of sequences of elements of Σ. For such a sequence $\alpha \in \Sigma^*$, $|\alpha|$ denotes the length of the sequence, for an index $i \in [1, |\alpha|]$, α_i denotes the i^{st} element in the sequence α. We also refer to sequences by extensionally listing their elements within angle brackets, as in $\alpha = \langle x_1, \ldots, x_n \rangle$.[2] ϵ denotes an empty sequence, as does $\langle \alpha_k, \ldots, \alpha_n \rangle$ whenever $n < k$. For two sequences α, β, $\alpha \leq \beta$ holds iff α is a prefix of β. Accordingly $\alpha < \beta$ holds iff $\alpha \leq \beta$ and $\alpha \neq \beta$. $\alpha \bullet \beta$ denotes the concatenation of the two sequences α and β ($\langle \alpha_1, \ldots, \alpha_n \rangle \bullet \langle \beta_1, \ldots, \beta_m \rangle = \langle \alpha_1, \ldots, \alpha_n, \beta_1, \ldots \beta_m \rangle$). Finally we adopt the convention that $\min_{x \in \emptyset} f(x) = +\infty$ for any real-valued function f of one variable, and $\arg\min_{x \in E} f(x)$ denotes the set $\{x \in E \mid f(x) = \min_{x' \in E} f(x')\}$.

[1] We occasionally drop the "dynamic" part, as dynamic oracles are a strict generalization of static ones.

[2] In this case, $|\alpha| = n$.

2.1 Active Imitation Learning

Imitation learning is concerned with using supervised feedback in order to learn models which can make sequential decisions. Example NLP problems for which imitation learning can be used are Named-Entity Recognition tagging (Brantley et al., 2020) and shift-reduce dependency parsing (Goldberg and Nivre, 2012). We focus on problems in which the model chooses from a fixed set of actions O at every step (also referred to as the *output lexicon*) and define an imitation learning input as follows:

Definition 1 (Imitation Learning Input)**.** An imitation learning input[3] x consists of a sequence $w = \langle w_1, \ldots, w_n \rangle$, a successor function $s : O^* \to \wp(O)$, and a stopping criterion $t : O^* \to \{\text{true}, \text{false}\}$.

Intuitively, the successor function s restricts the actions that the model can choose to the set $s(\alpha) \subseteq O$, depending on the sequence of previously taken actions α. Such a restriction is generally needed to ensure that only meaningful output (*e.g.* well-formed trees) are produced for a given input.

For our example of generating a sequence corresponding to a parse tree, the input sequence consists of word tokens. Our output lexicon consists of all possible word tokens that occur in the input, as well as opening brackets labeled with all possible nonterminals in the set N, e.g., NP(or S(, and the closing bracket). The stopping criterion is true once all tokens in the input have been generated in the output and there are no unmatched open brackets. For ease of presentation we will only consider context free parses without unary productions, i.e. we do not allow trees of the form $X(t)$ where t is any complete parse tree. This means the successor function allows generation of) whenever there is at least one more unmatched open bracket, the previous output is either a word token or another) and closing the bracket would not create a unary bracketing. s will allow opening brackets as long as there are more word tokens left to be produced than there are words left to produce and the last output was not a word token. Finally the s function allows w_i after an opening bracket or another word, if w_{i-1} has been produced.

We obtain a solution $\alpha_1, \ldots, \alpha_k$ for a given input with a *model* m by repeatedly choosing the next action α_k among the admissible actions in $s(\alpha_1, \ldots, \alpha_{k-1})$, according to the scores assigned by m. Whenever $t(\alpha_1, \ldots, \alpha_k)$ becomes true, the model will have to score the option of stopping against all possible outputs. This is relevant to problems such as machine translation, where it is possible to continue even after a potential stopping point. Our definition of an input restricts admissible candidate solutions to the set $\tau_x = \{\alpha \in O^* \mid \forall k \in [1, |\alpha|]\, \alpha_k \in s(\alpha_1, \ldots, \alpha_{k-1}) \wedge t(\alpha) = \text{true}\}$.

We assume that every imitation learning problem comes with a set Y of possible results, and that every (admissible) action sequence α for an input x can be interpreted as an element $[\![\alpha]\!] \in Y$. In our parsing example, the interpretation function simply takes a valid bracketing and maps it to the corresponding tree with Y being the set of parse trees for the sentence. Another example would be outputting the tokens of an SQL command and mapping them to their evaluation result relative to a database.

Note that $[\![\cdot]\!]$ is not necessarily an injective mapping. For the SQL example, different commands evaluate to the same results. In some settings the interpretation of an action sequences can depend on the words of the input sequence w, e.g., if our outputs were actions in classical shift-reduce parsing. For this reason, we assume a collection of interpretation functions indexed by the input rather than a unique, input-independent one. Finally, in order to unify the treatment of the training and test setting, we generally assume that there is a gold output $g \in Y$. This leads to this definition of an imitation learning problem:

Definition 2 (Imitation Learning Problem)**.** An imitation learning problem P is a set of *instances*, each being a triple $\langle x, g, [\![\cdot]\!] \rangle$ where x is an input, $g \in Y$ is a gold annotation. and $[\![\cdot]\!] : \tau_x \mapsto Y$ is a function interpreting any admissible output action sequence as an outcome in Y.

A model's performance is measured by a *loss function*. A loss function is a function $\mathcal{L} : Y \times Y \mapsto \mathbb{R}^+$. The arguments fed to the loss function typically are the interpretation of an action sequence and the gold annotation. For constituency parsing the loss function for training and testing is 1 minus the F1 score - for a loss function, smaller values should indicate better results. To give another example, for machine translation, a loss would be 1 minus the BLEU score computed between gold translations and the output translation. Because we measure the loss of an (admissible) output ac-

[3] We simplify to just input whenever the meaning is clear.

tion sequence α on a problem instance $\langle x, g, [\![\cdot]\!]\rangle$ through the quantity $\mathcal{L}([\![\alpha]\!], g)$, it is not necessary that the output action sequence and the gold annotation are of the same "type". The comparison is mediated by the interpretation function and training will aim to learn to produce action sequences that interpret to low loss targets.

2.2 Learning Set-Up

How does one learn in this setting? One option is to use reinforcement learning to obtain a model through trial and error feedback coming from the loss function (Sutton and Barto, 2018). This is generally not the most efficient way to use the information available. If it is possible to derive a sequence of outputs $\langle \alpha_1, \ldots, \alpha_m \rangle$ with minimum loss, then this can be used as the basis of standard imitation learning, without any exploration (Hussein et al., 2017). This is known as *static oracle learning* in NLP. In the parsing example this means obtaining the action sequence that is given in Fig. 1, as it corresponds to the "correct" parse tree, and training a classifier to produce $S($ as a first step given the input, then produce $NP($ given the input and $S($ and so on. We can further exploit the knowledge implicit in the loss function through active imitation learning(Hussein et al., 2017; Ross et al., 2011), also known as dynamic oracle learning in NLP (Goldberg and Nivre, 2012). In this setting the learning is active because it obtains feedback for which action is optimal for a given instance and partial action output $\alpha = \langle \alpha_1, \ldots, \alpha_k \rangle$, where we will call α a *prefix*. This makes it possible to learn to adjust for errors that a model is likely to make, and to explore different sequences of actions, in order to find one that is easy to learn.

When teaching a robot how to move, or learning to automatically drive, human intervention might be required in order to give the optimal action for every situation. We are focused on deriving optimal actions directly from gold outputs so that no further annotator intervention is necessary. We define a dynamic oracle as follows:

Definition 3 (Dynamic Oracle)**.** A *dynamic oracle* π for an imitation learning instance $\langle x, g, [\![\cdot]\!]\rangle$ and loss $\mathcal{L}$ is a function such that:

$$\forall \alpha \in O^* : \pi(\alpha) \in \arg\min_{o \in s(\alpha)} \min_{\beta \in \tau_x, \alpha \bullet o \leq \beta} \mathcal{L}([\![\beta]\!], g)$$

To put the definition of π in words: an oracle gives, for every prefix, an action that is the next step in a sequence that has the minimum loss possible for this prefix. Dynamic oracles enable the implementation of special learning algorithms with strong guarantees on test time performance and no exposure bias. One such algorithm is Dagger (Ross et al., 2011), which comes with attractive guarantees on model convergence. For clarity we provide the pseudo-code for Dagger, adjusted for our framing of the problem, in Figure 2, where we denote the *prediction of a model* ϕ for a given instance $in = \langle x, g, [\![\cdot]\!]\rangle$ and action sequence α as $\phi(\alpha, in)$.

- **Inputs**
  - interpolation schedule $\iota_0 \in (0, 1), \ldots$
  - instances $\langle x_1, g_1, [\![\cdot]\!]\rangle, \ldots, \langle x_n, g_n, [\![\cdot]\!]\rangle$
  - dynamic oracle π_j for each $\langle x_1, g_1, [\![\cdot]\!]\rangle$
  - starting model ϕ_0
- $Data = \emptyset$
- **Steps** for $i = 0$ to max steps:
  1. for $j = 1$ to n:
     (a) go to example $in = \langle x_j, g_j, [\![\cdot]\!]\rangle$ and set $\alpha \leftarrow \epsilon$
     (b) iterate:
        i. with probability ι_i set $pred = \phi_i(in, \alpha)$ otherwise set $pred = \pi_j(\alpha)$
        ii. add $\langle in, \alpha, \pi_j(\alpha)\rangle$ to $Data$
        iii. if $pred$ is to stop, end iterate
        iv. set $\alpha \leftarrow \alpha \bullet pred$
  2. train ϕ_{i+1} from $Data$

Figure 2: Pseudocode for Dagger.

The Dagger algorithm alternates between pursuing an optimal action and pursuing one chosen by the current model with probability ι_i. ι_0 is usually set to 0, to train a first model on optimal action sequences. By adding pairs of prefixes that a model visited and the dynamic oracle actions for these prefixes to the training data, models are able to learn what to do for prefixes they are likely to encounter. The last model trained is usually the one used at test time.

We presented dynamic oracles as the solution to an optimization problem over sequences. With this in mind, we will build on concepts from finite state automata in order to make these problems clearer and to solve them efficiently.

2.3 Finite State Machines

Given any finite set Q, called *states*, and finite set Σ, called *alphabet*, we call δ a transition function

if δ assigns a weight $w \in \mathbb{R} \cup \{+\infty\}$[4] to any triple $\langle q, o, q' \rangle \in Q \times \Sigma \times Q$. Given such a transition function, we write δ^* for the (weighted) transitive closure of δ. δ^* extends δ to words in Σ^* and is defined inductively:

$$\delta^*(q, \epsilon, q) = 0$$
$$\delta^*(q, \alpha \cdot o, q') = \min_{q''} \delta^*(q, \alpha, q'') + \delta(q'', o, q').$$

Where q, q', q'' range over Q, o over Σ and α over Σ^*, and all free variables are implicitly universally quantified.

In order to later discuss transducers, we will also use the classic extension of transition functions to a pair of a left-hand side alphabet and a right-hand-side alphabet $\langle \Sigma, \Lambda \rangle$. Such an (extended) transition function δ assigns a weight to any quadruple $\langle q, o_1, o_2, q' \rangle \in Q \times (\Sigma \cup \{\epsilon\}) \times (\Lambda \cup \{\epsilon\}) \times Q$.[5] In this extended case, the definition of the (weighted) transitive closure of δ is amended to:

$$\delta^*(q, \epsilon, \epsilon, q) = 0$$
$$\delta^*(q, \alpha, \beta, q') = \min_{\gamma, \lambda, o_1, o_2 \in H(\alpha, \beta)} \min_{q''} \delta^*(q, \gamma, \lambda, q'') + \delta(q'', o_1, o_2, q').$$

Where the first minimum is taken over the set $H(\alpha, \beta) = \{\langle \gamma, \lambda, o_1, o_2 \rangle \in \Sigma^* \times \Lambda^* \times (\Sigma \cup \{\epsilon\}) \times (\Lambda \cup \{\epsilon\}) \mid \langle o_1, o_2 \rangle \neq \langle \epsilon, \epsilon \rangle \text{ and } \langle \gamma \bullet o_1, \lambda \bullet o_2 \rangle = \langle \alpha, \beta \rangle\}$

This paper uses automata exclusively for minimum weight problems. This means that we only focus on tropical weighted finite state automata and transducers (Mohri, 2009), which use the addition and minimum operations. We drop both "tropical" and "weighted" where appropriate.

Definition 4 (Weighted Finite State Automaton)**.** A tropical weighted finite state automaton (automaton) A is a tuple $\langle q_0, Q, \Sigma, \delta, \rho \rangle$ where $q_0 \in Q$ is the start state, Q and Σ are the states and the alphabet, δ is a transition function and $\rho : Q \to \mathbb{R}$ is the final weight function. A defines a function $A(\alpha) = \min_{q \in Q} \delta^*(q_0, \alpha, q) + \rho(q)$. The weighted language $L(A)$ of A is the set $\{\langle \alpha, w \rangle \mid A(\alpha) = w\}$.

We say that a (non weighted) language $L \subseteq \Sigma^*$ is *regular*, iff there exists an automaton A_L such that, for any $\alpha \in \Sigma^*$, $A_L(\alpha) = 0$ if $\alpha \in L$ and $A_L(\alpha) = +\infty$ otherwise. Such an automaton is said to *recognize* L.

Definition 5. A tropical transducer T is a tuple $\langle q_0, Q, \Sigma, \Lambda, \delta, \rho \rangle$, where Σ and Λ are two alphabets, and δ is an extended transition function over theses two alphabets. All other members of T are exactly as in definition 4. T defines a weight function (of two arguments) $T(\alpha, \beta) = \min_{q \in Q} \delta^*(q_0, \alpha, \beta, q) + \rho(q)$. The weighted relation $L(T)$ of T is the set $\{\langle \langle \sigma, \sigma' \rangle, w \rangle \mid T(\sigma, \sigma') = w\}$

The size $|A|$ (resp. $|T|$) of an automaton A (resp. a transducer T) is defined as $|Q| + |\delta|$, where $|\delta|$ is the number of finite-weight transitions. If A and A' are both weighted automata, we write $L(A) \cap L(A')$ to denote the weighted language $\{\langle \alpha, w \rangle \mid w = A(\alpha) + A'(\alpha)\}$. This is the *intersection* of A and A'. If T is a transducer and A is an automaton, we write $L(T) \circ L(A)$ for the weighted relation $\{\langle \langle \alpha, \beta \rangle, w \rangle \mid w = T(\alpha, \beta) + A(\beta)\}$. Symetrically, we write $L(A) \circ L(T)$ for the weighted relation $\{\langle \langle \alpha, \beta \rangle, w \rangle \mid w = A(\alpha) + T(\alpha, \beta)\}$. Both are called *applications* of T to A. Note that the intersection of two automata can be expressed as an finite automaton as well and the application of a transducer can be expressed as another transducer:

Lemma 1. *1. If A and A' are automata, one can construct an automata $A \cap A'$ such that $L(A \cap A') = L(A) \cap L(A')$. Moreover, $A \cap A'$ can be computed in time $O(|A||A'|)$ (Rabin and Scott, 1959).*

2. If A is an automaton and T is a transducer, there exists a transducer $T \circ A$ such that $L(T \circ A) = L(T) \circ L(A)$. Moreover, $T \circ A$ can be effectively computed in time $O(|T||A|)$ (Mohri, 2004). The same holds, up to symmetry, for $L(A) \circ L(T)$, and we write $A \circ T$ for the corresponding transducer.

For all of the above statements the intuition is to construct a new automaton/transducer that has pairs of the two automata's states as its states and has a transition with weight $w + w'$ if the two automata have matching transitions with weight w and w' respectively. As a consequence, if A has m states, u transitions and A' (resp. T) has m' states, u' transitions, then $A \cap A'$ (resp. $A \circ T$ or $T \circ A$) has $O(|k||l|)$ states and $O(|u||u'|)$ transitions.

Definition 6. For any transducer T, we let V_T denote the function which maps every state to the

[4] As we will be reasoning about minimum weight paths, ∞ corresponds to an absent transition.

[5] Note the addition of the empty sequence ϵ to the left-hand-side and right-hand-side alphabets.

minimal score which can be assigned to some sequence read from that state. Formally: $V_T(q) = min_{\alpha,\beta,q'} \delta^*(q, \alpha, \beta, q') + \rho(q')$.

In other words, V_T gives the weight of the minimum weight or *shortest* path to any final state plus the weight of that state (Mohri, 2009).

Lemma 2. *For any transducer T and state q, the set*

$$\{\langle q, V_T(q)\rangle \mid q \in Q\}$$

can be computed in $O(|Q||\delta|)$ (Mohri, 2009).

This is a shortest path problem, solvable by the Bellman-Ford (Mohri, 2009) algorithm. If all weights are positive, this is amenable to $O(|Q|log(|Q|) + |\delta|)$ through Dijkstra's algorithm (Dijkstra, 1959).

3 Finite State Automata Oracles

We now provide generic dynamic oracles, prove the soundness of our constructions and provide complexity upper-bounds. We will encode all the possible action sequences for a given imitation learning problem as an automaton and then retrieve the next transition in a minimum loss complete solution for a given prefix.

The key question is how to derive an automaton of losses from a problem instance without having to explicitly go through all possible action sequences. In order to do this, we need three requirements guaranteeing applicability of finite-state techniques. First, we must be able to build a *decomposition automaton* inverting the interpretation function, and its language must not be empty. Second, we must be able to approximate the loss function with a transducer working over action sequences. Third, there must be an automaton recognizing the set of admissible candidate output action sequences for the considered input. These requirements are formally captured by the following definitions.

Definition 7. $\langle x, g, [\![\cdot]\!]\rangle$ has a *decomposable gold annotation* if the set $[\![g]\!]^{-1} = \{\alpha \in O^* \mid [\![\alpha]\!] = g\}$ is both regular and non-empty. An automaton recognising this set is called a *decomposition automaton*.

In our constituency parsing example, the decomposition automaton for a tree is simply the automaton that accepts the bracketing for the tree, e.g., the one recognizing "S(John VP(hit NP(the ball)))" for the tree in Fig. 1. The automaton recognizing this sequence would have positions in the gold output as states and would have transitions such as $1, John, 2$ or $2, VP(, 3$ with weight 0. For machine translation the decomposition automaton may recognize any of a number of possible gold translations. Note that our notion of "decomposable" is unrelated to the notion of "arc-decomposable" used in previous research on oracles for dependency parsing. Our notions of decomposability is concerned with decomposing every possible way of arriving at the gold output into a sequence of actions, while arc-decomposability tells us about the interaction between added edges during dependency parsing.

Definition 8. We say that $\mathcal{L}$ is *decomposable* if there exists a transducer T such that for any instance $\langle x, g, [\![\cdot]\!]\rangle \in P$ and sequences $\alpha, \alpha', \beta \in O^*$, if $\mathcal{L}([\![\alpha']\!], [\![\beta]\!]) < \mathcal{L}([\![\alpha]\!], [\![\beta]\!])$, then there exists $\beta' \in O^*$ such that $[\![\beta']\!] = [\![\beta]\!]$ and $T(\alpha', \beta') < T(\alpha, \beta)$.

Definition 8 relates the transducer and loss function with an inequality, not an equality. This provides more flexibility: we do **not** require that the loss function be directly computed by a transducer. If we did, then that would rule out very common losses such as F-Score. Rather, we allow transducers which conserve the right minima (see Lemma 3 below). Variations of the Levenshtein edit-distance between the output and gold-sequence are expressible as a (single state) transducer, and provides a generic loss function in practical cases. Consider our example of constituency parsing: the F-Score is the harmonic mean of two measures that require division by the total count of constituents present in a prediction and is hard to express as a transducer. However, as shown by Cross and Huang (2016), for purposes of an oracle, the number of incorrectly inserted and missing brackets (which corresponds to the edit distance between input and output for our setting) fits definition 8 and can thus replace a loss based on the F-Score. An incorrectly inserted bracket will always reduce precision without changing recall and vice versa for dropping a bracket. For our example problem the transducer would have have transitions $0, x, x, 0$ with weight 0, which map every symbol to itself, transitions $0, X(, \epsilon, 0,\ 0, \epsilon, X(, 0$ and $0, Y(, X(, 0$ which allow us to delete, insert, or relable any opening brackets $X(, Y($ with weight 1, as well as transitions $0, X), \epsilon, 0$ and $0, \epsilon, X), 0$ with weight 0.[6]

[6]Because closing brackets need to be matched, each incorrectly inserted one will incur a loss through an incorrect opening bracket

The next lemma states that for any set of action sequences, the input action sequence assigned minimum weight by some decomposition transducer is indeed a minimum loss action sequence out of that set.

Lemma 3. *Let $\mathcal{L}$ be decomposable, $\langle x, g, [\![\cdot]\!]\rangle \in P$, and T be as described in definition 8. For any subset $D \subseteq O^*$,*

$$\text{if } \tilde{\alpha} \in \arg\min_{\alpha \in D} \min_{\beta, [\![\beta]\!]=g} T(\alpha, \beta),$$
$$\text{then } \tilde{\alpha} \in \arg\min_{\alpha \in D} \mathcal{L}([\![\alpha]\!], g)$$

Proof. Let $\tilde{\alpha} \in \arg\min_{\alpha \in D} \min_{\beta, [\![\beta]\!]=g} T(\alpha, \beta)$ and $\tilde{\beta} \in \arg\min_{[\![\beta]\!]=g} T(\tilde{\alpha}, \beta)$. If there existed $\alpha' \in D$ such that $\mathcal{L}([\![\alpha']\!], g) < \mathcal{L}([\![\tilde{\alpha}]\!], g)$, it would follow from definition 8 that $\exists \beta', [\![\beta']\!] = [\![\tilde{\beta}]\!] = g$ and $T(\alpha', \beta') < T(\tilde{\alpha}, \tilde{\beta})$, which contradicts the definition of $\tilde{\alpha}$. Hence $\tilde{\alpha} \in \arg\min_{\alpha \in D} \mathcal{L}(\alpha, \beta)$. □

Finally, we need to add the regularity of the possible action sequences:

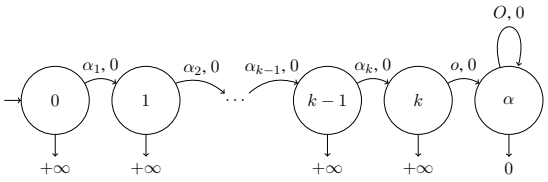
Definition 9. We say that x has *regular constraints* iff τ_x is regular.

For our context free parsing example, all the possible parses for a sentence can be expressed in a finite state automaton by virtue of the fact that any finite language is regular. We obtain an automaton of size $O(n^2)$ for an input of length n, when we construct states that encode how many brackets are open and which word was last produced. We would have a state $(3, c, t)$, which would be reached after producing, e.g., $((a($ for the input sentence $abcd$. t and f would be use to indicate whether we can still produce closing brackets before outputting the next word, to prevent outputs like $((a(b()$. We would allow, e.g., a transition of the form $(3, c, t),), (2, c, t)$. Note that we only need to maintain numbers up to the length of the inputs sentence, since no more can be used in a permissible parse for the input. Our construction for the action sequence automaton allows for unary bracketings, but since they do not occur in the gold output and would always incur additional loss, this will not constitute a difficulty.

From here on, we assume $\langle x, g, [\![\cdot]\!]\rangle$ to be an instance with both decomposable gold annotation and regular constraints, and $\mathcal{L}$ to be a decomposable loss function. We now proceed to a first oracle construction which follows naturally from these assumptions and some of the properties of finite state machines listed in section 2. Let $\alpha \in O^*$ represent a sequence of k actions that the model has already taken. We can find an optimal action for the next step: consider an arbitrary candidate action $o \in O$ and recall that the oracle must determine which, among the possible choices for o, is part of the minimum loss completion of α into an admissible action sequence. Letting (for any action sequence γ) $\mathsf{cont}_\gamma = \{\gamma' \in \tau_x \mid \gamma' \geq \gamma\}$ denote the set of admissible continuations of γ, we can formally rephrase our objective as choosing an action o minimizing the quantity $\min_{\alpha' \in \mathsf{cont}_{\alpha \bullet o}} \mathcal{L}([\![\alpha']\!], g)$.

To ease notational clutter, let us define $l(\alpha') = \min_{\beta' \in [\![g]\!]^{-1}} T(\alpha', \beta')$. Observe that lemma 3 ensures that we solve the oracle task if we find $\tilde{o} \in \arg\min_{o \in O} \min_{\alpha' \in \mathsf{cont}_{\alpha \cdot o}} l(\alpha')$. For if we find such $\tilde{o}$, letting $\tilde{\alpha} \in \arg\min_{\alpha' \in cont_{\alpha \bullet o}} l(\alpha')$, we have $\tilde{\alpha} \in \arg\min_{\alpha' \in \mathsf{cont}_\alpha} l(\alpha')$ (easily checked from the definition of $\tilde{o}$). Then, from lemma 3 follows that $\tilde{\alpha} \in \arg\min_{\alpha' \in \mathsf{cont}_\alpha} \mathcal{L}([\![\alpha']\!], g)$, which (since $\tilde{\alpha} \in \mathsf{cont}_{\alpha \bullet \tilde{o}}$) finally shows that $\tilde{o}$ is one of the optimal choices for continuing α, *i.e.* $\tilde{o} \in \arg\min_{o \in O} \min_{\alpha' \in \mathsf{cont}_{\alpha \bullet o}} \mathcal{L}([\![\alpha']\!], g)$.

How can we compute this quantity? We first build an automaton $A_{\alpha \bullet o}$ which recognizes the set $\alpha \bullet o \bullet O^*$. This is easily done as depicted below, with the following graphical conventions: states are circled, the start state (0) is marked with a left-dangling incoming arrow, arrows between states represented transitions, annotated with their label(s) and weight (a set of labels like O is a factored representation of one transition for each $o \in O$, all with the same indicated weight), and final weights are given by the downwards outgoing arrows.



In our example setting, this would be an automaton that accepts the brackets and word tokens produced so far, followed by all possible words and brackets and which assigns weight 0 to each of those transitions.

Let C_x be an automaton recognizing the set τ_x of admissible actions for x (C_x exists since x has regular constraints). In our example this would be an automaton that accepts all the valid bracketings of the input. Note that this can be represented as a finite state automaton, due to the limit on open

brackets we stipulated. Observe that $C_x \cap A_{\alpha\bullet o}$ (as provided by lemma 1) recognizes $\mathsf{cont}_{\alpha\bullet o}$.

Let now T be the transducer provided by definition 8, and $\mathcal{D}_g$ be a decomposition automaton for g. Consider the transducer $T_{\alpha\bullet o} = (C_x \cap A_{\alpha\bullet o}) \circ (T \circ \mathcal{D}_g)$ (provided by two successive applications of lemma 1), and let q_0 be its initial state. With a little work (we skip the details here, due to space limitations), one can show that if $\beta' \notin [\![g]\!]^{-1}$ or $\alpha' \notin \mathsf{cont}_{\alpha\bullet o}$ then $T_{\alpha\bullet o}(\alpha', \beta') = +\infty$, and that otherwise $T_{\alpha\bullet o}(\alpha', \beta') = T(\alpha', \beta')$. This guarantees:

$$\begin{aligned} &\min_{\alpha'} \min_{\beta'} T_{\alpha\bullet o}(\alpha', \beta') \\ &= \min_{\alpha' \in \mathsf{cont}_{\alpha\bullet o}} \min_{\beta' \in [\![g]\!]^{-1}} T(\alpha', \beta'). \end{aligned} \tag{1}$$

Recall that $V_{T_{\alpha\bullet o}}(q_0) = \min_{\alpha'} \min_{\beta'} T_{\alpha\bullet o}(\alpha', \beta')$. Equation (1) and preceding observations establish the soundness of the following oracle computation:

Oracle computation for prefix α. For each $o \in O$, construct $T_{\alpha\bullet o}$, then computes $V_{T_{\alpha\bullet o}}(q_0)$. Find and output the action o minimizing $V_{T_{\alpha\bullet o}}(q_0)$.

In terms of our example, this would mean taking the automaton that expresses all possible continuations of a partial parse and intersecting it with the automaton of all possible bracketings of the input. Then we apply a transducer that encodes the edit distance to the gold bracketing and extract the shortest path from the resulting automaton

We now briefly discuss the complexity of a single call to this oracle, and of a sequence of prediction, at each timestep of an input's processing. Recall that $k = |\alpha|$ and observe that $A_{\alpha\bullet o}$ has $O(k)$ states and $O(k)$ transitions. Let m_T, m_g and m_x denote the number of states of T, D_g and C_x resepectively, and e_T, e_g, e_x their respective number of finite-weight transitions. We consider the size of the alphabet O constant and exclude it from the underlying variables of all the asymptotic bounds reported. By lemma 1, computing $T_{\alpha\bullet o}$ is done in time $O(k|D_g||T||C_x|) = O(k(m_g + e_g)(m_x + e_x)(m_T + e_T))$, it has $O(km_gm_Tm_x)$ states and $O(ke_ge_Te_x)$ transitions. By lemma 2 computing $V_{T\alpha\bullet o}(q_0)$ is done in $O((km_gm_Tm_x)(ke_ge_Te_x))$, and is asymptotically the dominant term. Iterating (a constant number of times) over $o \in O$ leaves the asymptotic bound $O(k^2(m_gm_Tm_xe_ge_Te_x))$.

If a machine learning system builds and outputs a (complete) sequence of n actions in processing (entirely) a given input x, and needs to call the oracle at each timestep $k \in [1, n]$ (*i.e.*, there is a call on each prefix of length k of the complete action sequence), the overall cost of oracle calls in the processing of x will then be $O(n^3(m_gm_Tm_xe_ge_Te_x))$. If no negative weights are involved, this can be lowered to $O(n^2(m_gm_Tm_xlog(nm_gm_Tm_x) + e_ge_Te_x))$.

This is extremely suboptimal, because the algorithm discussed above is only *superficially* dynamic: at every timestep, an independent computation arises with redundant work all the way up to the prefix α of previous actions disregarding the result of previous timesteps' computations. In fact, shortest path computations can be performed in advance. To this aim, we can work with $T_\epsilon = C_x \circ (T \circ D_g)$, a transducer that combines the automaton of all possible input sequences with the decompositions of the gold output. This transducer is only dependent on the problem and the loss function, hence only needs to be computed once, at the time the corpus is created. We can use the following observation: when we start producing the output sequence, the best action is the first action of the best path from T_ϵ's start state q_0. After outputting an action a, we can obtain a set c of states in T_ϵ that are reachable by reading a from q_0. The best next action must then be the first action of some path from some state $q' \in c$, which is determined according to the cost of reaching q' through a plus the weight of the best path from q'. This updating can be carried forward during the whole decoding process. This frees us from having to repeat a lot of computation, as we will see that we only need to compute the best paths in T_ϵ once. To formalize this: let $T_\epsilon = \langle q_0, Q, O, O, \delta, \rho \rangle$, then

$$\begin{aligned} &\min_{\alpha' \in \mathsf{cont}_\alpha} \min_{\beta' \in O^*} T_\epsilon(\alpha', \beta') \\ &= \min_{q'}(\min_{\beta} \delta^*(q_0, \alpha, \beta, q') + V_{T_\epsilon}(q')). \end{aligned} \tag{2}$$

Let $\mathsf{Pre}_\alpha(q') = \min_{\beta \in O^*} \delta^*(q_0, \alpha, \beta, q')$, the minimum weight of a path reaching q' from the start state with α. Using Eq. (2), we have

$$\begin{aligned} &\arg\min_{o \in O} \min_{\alpha' \in \mathsf{cont}_{\alpha\bullet o}} \min_{\beta'} T_\epsilon(\alpha', \beta') \\ &= \arg\min_{o} \min_{q'} \mathsf{Pre}_{\alpha\bullet o}(q') + V_{T_\epsilon}(q'). \end{aligned}$$

Finally, since by construction $T_\epsilon(\alpha', \beta') \neq +\infty$ entails $[\![\beta']\!] = g$, it is sufficient to find

$$\tilde{o} \in \arg\min_{o} \min_{q'}(\mathsf{Pre}_{\alpha\bullet o}(q') + V_{T_\epsilon}(q')) \tag{3}$$

to solve the oracle problem for prefix α. Our second construction thus proceeds as follows:

Second oracle computation for α. When constructing the problem, compute V_{T_ϵ} for each instance. During iteration, to obtain the optimal next action $o \in O$ for prefix α, choose the minimum of $\mathsf{Pre}_{\alpha \bullet o}(q') + V_{T_\epsilon}(q')$.

What we gain from this obviously depends on the cost of computing $\mathsf{Pre}_\alpha(q')$ for every state q'. The key insight is that $\mathsf{Pre}_{\alpha \bullet o}$ can be computed inductively from Pre_α:

$$
\begin{aligned}
&\mathsf{Pre}_{\alpha \bullet o}(q') \\
&= \min_{\beta \in O^*} \delta^*(q_0, \alpha \bullet o, \beta, q') \\
&= \min_{\beta', \beta'', p} \min_{q''} \delta^*(q_0, \alpha, \beta', q'') \\
&\qquad + \min_{q'''} \delta(q'', o, p, q''') \\
&\qquad\quad + \delta^*(q''', \epsilon, \beta'', q') \\
&= \min_{q''} \mathsf{Pre}_\alpha(q'') \\
&\qquad + \min_{q'''} \min_{p} \delta(q'', o, p, q''') \\
&\qquad\quad + \delta^*(q''', \epsilon, \beta'', q')
\end{aligned}
$$

Obverse that in the second equality above, the quantity $\min_{q'''} \min_p \delta(q'', o, p, q''') + \delta^*(q''', \epsilon, \beta'', q')$ depends only on T_ϵ, the states q'' and q', and **not** on the prefix α. We thus refer to this quantity as $\mathcal{C}(q'', q')$. Since it does not depend on α, it can be precomputed once for every pair $\langle q'', q' \rangle$, and reused through every iteration. The cost of this precomputation is asymptotically bounded by $O(|Q|^3)$: the lion's share is computing a table for the (lhs) epsilon closure $\delta^*(q''', \epsilon, \beta'', q')$, for all pairs $\langle q''', q' \rangle$. This is an instance of an all-pair shortest-path problem and solved with the Floyd-Warshall algorithm (Mohri, 2009). This is also akin to considering T_ϵ has an automaton rather than a transducer, using only the 'input' side (lhs) of transitions, and eliminating ϵ transitions. Note in passing, that computing Pre_ϵ is a similar problem and therefore done with the same asymptotic bound.

Because $\mathsf{Pre}_{\alpha \bullet o}$ can be computed inductively, it is possible to update it as the Dagger algorithm is going through a problem instance computing first Pre_ϵ, and then updating by taking a minimum over all possible transitions for the next action produced. The induction step then computes $\mathsf{Pre}_{\alpha \bullet o}(q')$ from the different $\mathsf{Pre}_\alpha(q'')$ and $\mathcal{C}(q'', q')$ in $O(|Q|^2)$ (since for each entry q' we need to range over all q''). This could be reduced to constant time by looking just at the inputs of T_ϵ and making the induced automaton deterministic (Hopcroft et al., 2006), however, that would come at the cost of a worst-case exponentially larger precomputation.

We now turn to the complexity analysis of the refined oracle construction. Computing V_{T_ϵ} is the only precomputation that we have not yet addressed, and can be achieved in time $O(|Q||\delta|)$ before any Dagger iterations. We can bound this with $O(|Q|^3)$ as well. Oracle calls will receive a (possibly empty) prefix of the form α. To compute the oracle we have to compute $\mathsf{Pre}_{\alpha \cdot o}$ from Pre_α, which will cost $O(|Q|^2)$ for a nondeterministic automaton, and then compute the minimum in equation (3). Hence, over a sequence of n actions with oracle calls in one entire pass over input x in Dagger, the total cost of oracle calls will be bounded by $O(|Q|^3 + n|Q|^2)$, or with the same notations as before $O((m_x m_g m_T)^3 + n(m_x m_g m_T)^2)$. Holding other parameters fixed, our second construction is much more efficient with respect to n, the length of the input. We can therefore conclude that our second construction is much more efficient during the actual Dagger training steps, with most of the computation moved into preprocessing, which is only needed once during the lifetime of a corpus.

Applied to our context free parsing example, our analysis bounds an oracle call (and update of Pre) with $O(n^6)$. The precomputation is bounded by $O(n^9)$, and the total of oracle calls in building a complete output is bounded with $O(n^7)$. However, we used here a generic bound that will hold for all instances of our method. No assumption were made on the nature of the automata involved[7], and specific instances will allow more efficient implementation of automata-theoretic operations, with no change to the general framework. We leave discussion of the automata theoretic properties that enable more efficient oracles for future work.

4 Related Work

Our dynamic oracle construction is general and only takes limited bookkeeping as seen in the previous section. However, this computation can still be costly and one topic in existing imitation learning is avoiding oracles computations where possible. A recent approach uses statistical techniques for the so called task of *apple tasting* to learn when it is necessary to call to an expensive oracle and when it is possible to instead use a cheap heuristic (Brantley et al., 2020). We could implement trans-

[7] Also, we relied for simplicity on a worst-case estimation $O(Q^2)$ for the number of transitions of T_ϵ.

ducer application and other computations on the finite state automata in a lazy manner and avoid oracle computations in order to save on computations for the automata and for tracking shortest paths. Other work uses reinforcement learning in order to derive approximate oracles (Yu et al., 2018; Fried and Klein, 2018). By using reinforcement learning to replace many of the oracle evaluation, it would also be possible to save on automaton construction.

The finite state approach we have sketched can give us an optimal action to take next. While this is sufficient to implement active imitation learning with a technique like Dagger, in some settings it can be beneficial to use not just information on what the best next action is, but rather to obtain the minimum loss for every available action in a given state (Ross and Bagnell, 2014) and to then train a loss aware classifier. As we are already computing these quantities, our algorithms would also suitable for this setting.

Related Dynamic Oracles There are a number of previously published oracles that are related to our setting, even if they proceed slightly differently. They are particularly used for dependency or constituency parsing. Note that our approach for constructing an automaton expressing the loss for all possible continuations could be applied to setting where the output is produced in a fashion other than left to right. Assume, e.g., that we use an algorithm which produces parse trees by "splitting" a sentence into sub-sequences repeatedly as in Stern et al. (2017). Let the output generated to far be S(NP(The old baker)VP(uses a sharp knife)). We would then simply construct an automaton equivalent to the regular expression S(NP(.∗The.∗old.∗ $baker$.∗)VP(.∗uses.∗a.∗sharp.∗knife.∗)), where .∗ stands for an arbitrary sequence of brackets. Through application of the loss transducer approximating the loss for all possible action sequences, we could retrieve the minimum loss continuation. We leave work on whether this construction allows for more efficient look-up to future work.

A predecessor of the work by Stern et al. (2017) is the paper by (Cross and Huang, 2016) which discusses a shift-reduce system for constituency parsing and gives a constant time dynamic oracle for this system. It would be possible to express their setting, as well as those of Coavoux and Crabbé (2016), Fernández-González and Gómez-Rodríguez (2018b) and the discourse parsing focused on of Hung et al. (2020) in our framework.

Dynamic oracles have also been developed for different formalizations of the dependency parsing problem (Goldberg and Nivre, 2012; Goldberg et al., 2014) for shift reduce parsing. For the projective setting, one could generalize these oracles by translating dependency tree to constituency trees Mareček and Žabokrtský (2011) or tag sequences (Gómez-Rodríguez et al., 2020). Oracles for non-projective dependency and constituency parsing (Coavoux and Cohen, 2019; Nederhof, 2021; Gómez-Rodríguez and Fernández-González, 2015; Fernández-González and Gómez-Rodríguez, 2018a; de Lhoneux et al., 2017; Gómez-Rodríguez et al., 2014) can in certain cases be computed in polynomial time, but would be harder to express in this framework without necessitating extremely large automata as it would be difficult to encode the different admissible sets of actions.

Our idea of using interpretations of action sequences is inspired by Interpreted Regular Tree Grammars (IRTGs) (Koller and Kuhlmann, 2011). Our approach works in terms of automata over string sequences and IRTGs are based on automata over trees. In future work we will use IRTGs to extend our approach to additional domains.

5 Conclusion

This paper gives a generic approach for deriving dynamic oracles for NLP. The oracles make it possible to implement error aware learning and learning in ambiguous environments for a wide range of NLP problems, including most problems that can be approached with sequence to sequence models. There is no need to derive new oracles for every new loss or set of output actions, instead automata can be derived once and reused if only part of a problem changes. We also showed how to substantially improve the efficiency of oracle lookup, by moving most computational cost into a one time pre-computation.

References

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010, Austin, Texas. Association for Computational Linguistics.

Kianté Brantley, Hal Daumé III, and Amr Sharaf. 2020. Active imitation learning with noisy guidance. In

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2093–2105, Online. Association for Computational Linguistics.

Maximin Coavoux and Shay B. Cohen. 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 204–217, Minneapolis, Minnesota. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 2016. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 172–182, Berlin, Germany. Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018a. A dynamic oracle for linear-time 2-planar dependency parsing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 386–392, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018b. Dynamic oracles for top-down and in-order shift-reduce constituent parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1303–1313, Brussels, Belgium. Association for Computational Linguistics.

Daniel Fried and Dan Klein. 2018. Policy gradient as a proxy for dynamic oracles in constituency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 469–476, Melbourne, Australia. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha, Qatar. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Michalina Strzyz, and David Vilares. 2020. A unifying theory of transition-based and sequence labeling parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3776–3793, Barcelona, Spain (Online). International Committee on Computational Linguistics.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.

Shyh-Shiun Hung, Hen-Hsen Huang, and Hsin-Hsi Chen. 2020. A complete shift-reduce Chinese discourse parser with robust dynamic oracle. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 133–138, Online. Association for Computational Linguistics.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2).

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 2–13, Dublin, Ireland. Association for Computational Linguistics.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104, Pisa, Italy. Association for Computational Linguistics.

David Mareček and Zdeněk Žabokrtský. 2011. Gibbs sampling with treeness constraint in unsupervised dependency parsing. In *Proceedings of Workshop on Robust Unsupervised and Semisupervised Methods in Natural Language Processing*, pages 1–8.

Mehryar Mohri. 2004. *Weighted Finite-State Transducer Algorithms. An Overview*, pages 551–563. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mark-Jan Nederhof. 2021. Calculating the optimal step of arc-eager parsing for non-projective trees. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2273–2283, Online. Association for Computational Linguistics.

M. O. Rabin and D. Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Xiang Yu, Ngoc Thang Vu, and Jonas Kuhn. 2018. Approximate dynamic oracle for dependency parsing with reinforcement learning. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 183–191, Brussels, Belgium. Association for Computational Linguistics.