Multi-Task Structured Prediction for Entity Analysis: Search-Based Learning Algorithms (Supplementary Material)

Chao Ma MACHAO@EECS.OREGONSTATE.EDU
Janardhan Rao Doppa† JANA@EECS.WSU.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
Hamed Shahbazi SHAHBAZH@EECS.OREGONSTATE.EDU
Xiaoli Fern XFERN@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University, Corvallis, OR 97331, USA
†*School of EECS, Washington State University, Pullman, WA 99164, USA*

Editors: Yung-Kyun Noh and Min-Ling Zhang

Appendix A.

A.1. Beam Search Inference Algorithm

Algorithm 1 is the beam search inference algorithm for structured prediction. The beam search explores the search space guided by a scoring function of the form $w \cdot \Phi(x, y)$ until it reaches a locally optimal state with a corresponding output $\hat{y}$. Each search step picks the best node in the beam according to the scoring function, and generates all its successors obtained by changing some output variable y_i^j of the parent state to a different value in the domain of that variable. It replaces the expanded node in the beam with the successor states, sorts the beam by the scoring function, and then drops all nodes beyond the beam width.

Algorithm 1 Beam Search Inference

```
Input: $x$: structured input, $\Phi(x, y)$: joint feature function, $(I, \texttt{Succ})$: search space definition,
$b$: beam width, $w$: weights of features
Output: $\hat{y}$, the best scoring output
1: Initialization: $y_0 = I(x)$ and $Beam \leftarrow \{y_0\}$
2: repeat
3:     $\hat{y} \leftarrow \arg\max_{y \in Beam} w \cdot \Phi(x, y)$ // Selection
4:     $Beam \leftarrow Beam \setminus \{\hat{y}\}$ // Remove the node selected for expansion
5:     $Candidates \leftarrow Beam \cup \texttt{Succ}(\hat{y})$ // Expansion
6:     $Beam \leftarrow$ Top-$b$ scoring outputs in $Candidates$ // Pruning
7: until max steps or local optima is found
8: return best scoring output $\hat{y}$
```

A.2. Pruning Algorithms

Here we provide the pseudocode for the two pruning strategies described in Section 4.3.

Algorithm 2 Score-Agnostic Pruning Function Learning

Input: D: training examples, α: pruning parameter
Output: P, action pruning function

```
 1: Initialization: $\mathcal{R} \leftarrow \emptyset$
 2: for $i = 1$ to $MAX$ do
 3:     for each training example $(x, y^*)$ do
 4:         if $i == 1$ then
 5:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{\ (GOOD(I) > BAD(I))\ \}$
 6:         else
 7:             $\mathcal{A}_P \leftarrow$ Top-$\alpha\,|\mathcal{A}(I)|$ actions from $\mathcal{A}(I)$ scored using $P$ // pruning action space
 8:             $M \leftarrow GOOD(I) \cap \mathcal{A}_P^-$   // where we define $\mathcal{A}_P^- = \mathcal{A}(I) \setminus \mathcal{A}_P$
 9:             if $M$ is not empty then
10:                 $M' \leftarrow$ Top-$|M|$ actions from $BAD(I) \cap \mathcal{A}_P$ scored using $P$
11:                 $\mathcal{R} \leftarrow \mathcal{R} \cup \{\ (M > M')\ \}$
12:             end if
13:         end if
14:     end for
15:     $P \leftarrow$ Rank-Learner($\mathcal{R}$)
16: end for
17: return $P$
```

Algorithm 3 Score-Sensitive Pruning Function Learning

Input: D: training examples, α: pruning parameter, w: weights of the scoring function
Output: P: action pruning function

```
 1: $P_0 \leftarrow$ Random function
 2: for $i = 1$ to $MAX$ do
 3:     Initialization: $\mathcal{R} \leftarrow \emptyset$
 4:     for each training example $(x, y^*)$ do
 5:         $\mathcal{A}_{P_{i-1}} \leftarrow$ Top-$\alpha\,|\mathcal{A}(I)|$ actions from $\mathcal{A}(I)$ scored using $P_{i-1}$ // pruning action space
 6:         $\hat{y}_i \leftarrow$ Beam-Search-Inference($x, w, \mathcal{A}_{P_{i-1}}$) // do structured prediction
 7:         $M_i \subset \mathcal{A}(I)$ is the set of actions corresponding to the mistakes in $\hat{y}_i$
 8:         if $M_i$ is not empty then
 9:             $M_i' \leftarrow \bigcup_{j=1}^{i} M_j$ // avoid these actions
10:             $PREF \leftarrow GOOD(I) \cup$ Top-($|\mathcal{A}_{P_{i-1}}| - |GOOD(I)|$) actions from $BAD(I) \setminus M_i'$ scored by $P_{i-1}$
11:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{\ (PREF > M_i')\ \}$
12:         end if
13:     end for
14:     $P_i \leftarrow$ Rank-Learner($\mathcal{R}$)
15: end for
16: return $P_{MAX}$
```

A.3. Testing Accuracy of Cyclic Algorithms

In this section, we present a study of the testing accuracy with different test cycles for the two different cyclic algorithms: `Unshared-Wt-Cyclic` and `Shared-Wt-Cyclic`. Figure 1 shows Hamming accuracy of the ACE05 development set w.r.t. the number of testing cycles for the two cyclic algorithms. The left figure is for `Unshared-Wght-Cyclic`, and the right is for `Shared-Wght-Cyclic`. The current model we use for testing is trained using 10 training cycles. In each figure, each curve corresponds to one task ordering. The curves shows that, irrespective of the task ordering, the best testing accuracy can be reached after 3 to 4 testings cycles, and is stable afterwards. Since there is no accuracy loss for increasing the number of cycles, we can do as many cycles as there is time for. In our experiments, we stopped the iterations once more than 95% of the predicted outputs did not change in the previous two cycles.

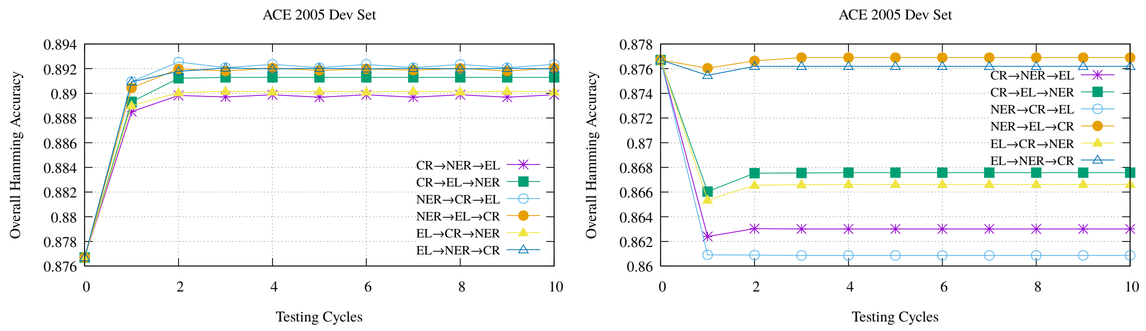

Figure 1: Hamming accuracy on ACE05 Dev w.r.t. testing cycles for Unshared-Wght-Cyclic (left) and Shared-Wght-Cyclic (right).