JMLR: Workshop and Conference Proceedings 80:1–16, 2017 ACML 2017

Multi-Task Structured Prediction for Entity Analysis: Search-Based Learning Algorithms

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

Abstract

Entity analysis in natural language processing involves solving multiple structured prediction problems such as mention detection, coreference resolution, and entity linking. We explore the space of search-based learning approaches to solve the problem of *multi-task structured prediction* (MTSP) in the context of entity analysis. In this paper, we study three different search architectures to solve MTSP problems that make different tradeoffs between speed and accuracy of training and inference. In all three architectures, we learn one or more scoring functions that employ both intra-task and inter-task features. In the "pipeline" architecture, which is the fastest, we solve different tasks one after another in a pipelined fashion. In the "joint" architecture, which is the most expensive, we formulate MTSP as a single-task structured prediction, and search the joint space of multi-task structured outputs. To improve the speed of joint architecture, we introduce two different pruning methods and associated learning techniques. In the intermediate "cyclic" architecture, we cycle through the tasks multiple times in sequence until there is no performance improvement. Results on two benchmark domains show that the joint architecture improves over the pipeline approach as well as the previous state-of-the-art approach based on graphical models. The cyclic architecture is faster than the joint approach and achieves competitive performance.

Keywords: Structured Prediction, Natural Language Processing, Learning for Search

1. Introduction

Many problems in AI including natural language processing (NLP) and computer vision require solving multiple related structured prediction tasks. Entity Analysis is one of the key steps in NLP and includes multiple subtasks such as detecting the mentions, clustering them to corefering sets, linking them to entities, and identifying their semantic roles. Each task requires jointly assigning values to multiple inter-dependent output variables.

In *multi-task structured prediction* (MTSP), we learn a single joint scoring function to evaluate candidate outputs of all tasks. The scoring function includes inter-task and intra-task features and is trained with the goal of scoring the correct outputs of all tasks higher than all alternatives. Learning the scoring function involves adjusting its weights to make

it consistent with the training data. Given such a scoring function, the inference task is to generate the outputs for all tasks that maximizes the joint scoring function. By viewing MTSP problem through the lens of AI search, we design learning algorithms and heuristics to search the space of solutions to find the best scoring output.

Two different architectures for MTSP present themselves as natural candidates. One is a "*pipeline*" architecture, where the different tasks are solved one after another in sequence. Each task in the pipeline adds more information that is used by the following tasks. While it has the advantages of simplicity and reduced search space, as we will see, the pipeline architecture is too sensitive to the task order and is prone to error propagation.

The second natural candidate is a "*joint*" architecture, where we treat the MTSP problem as a single task and search the joint space of multi-task structured outputs. Although it offers an elegant unified framework, the joint architecture poses multiple challenges. First, the branching factor of the joint search space increases in proportion to the number of tasks, making the search too expensive. To address this problem and make the training process more efficient, we learn a pruning function that prunes bad candidate solutions from the search space. Second, it is also important to initiate the search from a good starting solution to reduce the effective depth of the search. We do this by training an i.i.d. classifier to predict each output separately. Given a good initializer, it may only be necessary to correct a few mistakes, which reduces the effective search depth. Third, even with reduced branching factors and depth, exhaustive search is impractical. Following many other works, we employ best first beam search, which is space efficient.

Finally, we introduce a third search architecture referred as "*cyclic*," whose complexity is intermediate between the above two architectures. The different tasks are done in a sequence, but repeated in the same order as long as the performance as indicated by the current task's scoring function improves. The cyclic architecture has the advantage of not increasing the branching factor of the search beyond that of a single task, while offering some error tolerance and robustness with respect to task order. We make the following contributions in this paper. First, we establish the viability of search-based multi-task structured prediction for entity analysis by jointly solving named entity recognition, coreference, and entity linking tasks on multiple benchmark datasets, namely ACE 2005 (25) and TAC-KBP 2015 (17). Second, we show that the joint approach not only outperforms the pipeline approach with all task orders, but also the prior state-of-the-art approach based on graphical models. Third, we develop and evaluate new search space pruning approaches. The *score-agnostic pruning* method, which prunes the search space before learning the scoring function, reduces the inference time by about half with negligible loss in accuracy. The *score-sensitive pruning* approach learns a pruning function after the scoring function has been learned and improves the accuracy further. Finally, we show that the cyclic architecture offers competitive performance compared to the joint architecture at a reduced computational cost even relative to the pruning-based approaches.

2. Related Work

Prior work on structured prediction mostly considers single tasks. There are many frameworks to solve structured prediction problems with varying strengths and weaknesses. They include generalization of standard classification approaches such as conditional random field (CRF) (19), structured SVM (SSVM) (32), and structured perceptron, which require a good

inference algorithm to make predictions; and search-based approaches that learn different forms of search control knowledge such as greedy policies (9; 29), heuristic functions (8; 35), heuristic and cost functions (11; 20), and coarse-to-fine knowledge (34).

There is some work on jointly solving two structured prediction tasks (7; 14; 15; 22), but very little work on jointly learning and reasoning with three or more tasks (31; 13). There are graphical modeling approaches that learn a global scoring function in the framework of CRFs or Structured SVMs (14; 10; 31; 13). Indeed, our work is inspired by the work of (13) which employed graphical models and approximate inference via belief propagation for integrating multiple NLP subtasks for joint entity analysis. Integer linear programming (ILP) (30) formulation and inference is another potential approach, and has shown a lot of success in practice (10; 5). But it faces severe efficiency issue due to the large number of variables and constraints from multiple tasks. Also, the ILP formulation leads one to use optimized blackbox ILP engines, where learning new search control knowledge, e.g., the pruning rules in our approach, is difficult.

In this paper, we address the problem of joint inference and learning through the framework of search-based structured prediction, which combines the benefits of structured SVM and AI search, and has found success in multiple applications (8; 35; 9; 11). Some approaches learn a scoring function to guide beam search in the space of *partial* structured outputs (incremental prediction approach) (7; 2; 16). In contrast, we perform search in the space of *complete* structured outputs (12) and use good initialization to improve the accuracy of learning and inference. Our formulation also allows us to optimize non-decomposable loss functions. Additionally, we also handle latent variables, which do not appear in the supervised output, but nevertheless indirectly determine the output. One example of latent variables are the coreference links between a mention and a single previous (parent) mention that it co-refers. While there are many potential parents for a mention, they are not usually provided in the supervised output, but are extremely useful.

3. Problem Setup

Multi-Task Structured Prediction. We consider the problem of multi-task structured prediction (MTSP), where the goal is to predict the structured outputs of k ($k > 1$) related tasks, $y = (y^1, y^2, \cdots, y^k)$, for a given structured input x. Without loss of generality, assume that the structured output for a task t, y^t consists of T output variables: $y^t = (y^t_1, y^t_2, \ldots, y^t_T)$, and each output variable y^t_j can take values from a candidate set $C(y^t_j)$ of size d. We are provided with a training set of input-output pairs $\{(x, y^*)\}$, where x is a structured input and $y^* = (y^{1*}, y^{2*}, \cdots, y^{k*})$ is the correct multi-task structured output. The goal is to learn a function/predictor that can accurately map structured inputs to multi-task structured outputs.

Single task structured prediction problems are traditionally formulated as learning a linear scoring function of *a joint feature vector* Φ over an input and candidate output pair x, y so that for any input x, the correct output y^* has the highest score over all possible y's. We generalize this to multi-task structured prediction, where Φ now consists of both intra-task and inter-task features, which respectively encode the dependencies between the output variables of a single task and different tasks.

MTSP for Entity Analysis. In this paper, we consider *entity analysis* which consists of several related NLP tasks in recognizing and mapping noun phrases, also called mentions,

to entities in a knowledge base (KB). In particular, these include named entity recognition (NER), coreference resolution (CR), and entity linking (EL). NER refers to tagging named entities with their semantic types, while CR and EL respectively refer to clustering coreferant mentions and linking them to the corresponding KB entries (26; 17; 27). The strong interdependence between these tasks can be seen in the following example:

> "He left [*Columbia*] in 1983, ... after graduating from [*Columbia University*], he worked as a community organizer in Chicago ..."

In this example, while it is difficult to identify the meanings of the first instance of [*Columbia*] in isolation, it is straightforward to infer it from the second mention, once we have coreference information between the mentions, which follows from their proximity. In general, the three tasks can benefit by mutually constraining each other, a fact that has been established in prior work through joint graphical modeling (13).

To simplify the entity analysis problem and make it easier to compare to prior work, we assume the availability of extracted mentions for each document and all three tasks are applied to the same sequence of mentions as input. The output of the three tasks has the same size, which equals to the number of mentions. Note that our framework allows to include mention extraction as another task, although it significantly increases the size of the multi-task output space.

Given a document x that consists of T mentions m_1, m_2, ..., m_T in the textual order, we use y^c to denote the coreference resolution output, y^n to denote the named entity typing output, and y^l to denote the linking output. The entity analysis output can be written as $y = (y^c, y^n, y^l)$. For each sub-structure output y^t, the interpretation of decision on a mention m_i, denoted by y^t_i, is as follows:

- Coreference decision $y^c_i \in \{1 \dots i\}$ represents an antecedent mention index $j \leq i$ where m_i is coreferent to m_j. When $j = i$, m_i starts a new singleton cluster. Thus, each coreference output forms a left-linking tree. Note that the left-linking tree is not unique for a given coreference clustering.
- Entity Typing decision $y^n_i \in \mathbf{T}$ is a semantic tag assigned to m_i ($\mathbf{T}$ is a constant set).
- Entity Linking decision y^l_i is a knowledge base entry e, from a heuristically generated candidate set.[1]

In MTSP, we seek to learn an output scoring function $S(x, y)$ which takes the form $S(x, y) = w \cdot \Phi(x, y)$. The joint input-output feature vector $\Phi(x, y)$ can be decomposed into two groups: intra-task features, and inter-task features:

$$\Phi(x,y) = \underbrace{\Phi_1(x,y_1) \circ \cdots \circ \Phi_k(x,y_k)}_{\text{intra-task features}} \circ \underbrace{\cdots \circ \Phi_{(t_i,t_j)}(x,y_{t_i},y_{t_j}) \circ \cdots}_{\text{1st-order inter-task features}} \circ \underbrace{\cdots}_{\text{higher-order features}} \quad (1)$$

where $\Phi_t(x, y_t)$ denotes intra-task features for the t^{th} task, $\Phi_{(t_i,t_j)}(x, y_{t_i}, y_{t_j})$ denotes the first-order inter-task features for tasks t_i and t_j, and $\circ$ stands for concatenation of features. Note that we can easily add higher-order inter-task features as needed.

1. In some datasets, we also employ a slightly different definition $y^l_i = (q, e)$, where the query q is a sub-span of m_i, and e is the KB entry that can be retrieved using q as keyword. A value (q, e) of y^l_i is correct if $e = e^*_i$. Since there could be more than one q that can link to the same e, under this definition, y^{l*} would also be non-unique.

For our entity analysis problem, we employ c, n, and l to represent coreference, NER typing, and linking tasks respectively. For the intra-task group, we aggregate the feature vectors for individual tasks: $\Phi_c(x, y_c) \circ \Phi_n(x, y_n) \circ \Phi_l(x, y_l)$. For the inter-task group, we only employ the first-order inter-task features, and aggregate the feature vectors for all task pairs: $\Phi_{(c,n)}(x, y_c, y_n) \circ \Phi_{(c,l)}(x, y_c, y_l) \circ \Phi_{(n,l)}(x, y_n, y_l)$.

4. Search-based Learning Algorithms for MTSP

In this section, we present three different search architectures, *pipeline, joint*, and *cyclic*, with varying speed and accuracy tradeoffs for solving MTSP problems. We first describe the details of structured SVM training and search-based inference that is common to all three approaches, and subsequently, explain the three architectures.

4.1. Structured SVM Training and Search-based Inference

The key idea is to learn a function to score the candidate outputs generated by beam search. We employ SSVM approach for learning the scoring function due to its robustness (18). The main advantages of search-based inference and training over graphical model approach are: **1)** enable the injection of procedural knowledge through the design of appropriate search space; **2)** delink the complexity of features from the computational complexity of inference; and **3)** let the user control the time for inference based on the application needs.

4.1.1 Structured SVM Training.

Structured SVMs are a generalization of SVMs for standard classification. They learn a scoring function of the form $w \cdot \Phi(x, y)$ in order to rank the correct output y^* above exponentially many alternative candidate outputs $y \in Y(x) \setminus y^*$ for a training input x. SSVM employs the iterative cutting plane algorithm to efficiently solve this optimization problem. The key idea is to maintain a small set of active constraints A for tractability. It performs the following two steps in each iteration: 1) Solves the optimization problem with constraints A; and 2) Adds a most violated constraint for each training example to A. The training algorithm stops when no more constraints can be added to A. To compute the most violated constraint for a training input x, we need to search the space of candidate outputs $Y(x)$ to find the best scoring output $\hat{y}$. We employ beam search for this task.

Algorithm 1 Structured SVM Training

Input: D, training examples,
$\Phi(x, y)$, joint feature function
Output: w, the scoring function weights

Initialization: active constraint set $A \leftarrow \emptyset$
repeat
  $w \leftarrow$ batch optimization with constraints A
  for each training example $(x, y^*) \in D$ **do**
    $\hat{y} \leftarrow$ Beam-Search-Inference(x, w)
    if $\hat{y} \neq y^*$ **then**
      Add constraint $w \cdot \Phi(x, y^*) > w \cdot \Phi(x, \hat{y})$ to A
    end if
  end for
until convergence
return weights of the scoring function w

Latent SSVM. Some MTSP problems have hidden structure which is not apparent in the inputs and outputs, and may be represented by the latent variables h. For example, in coreference resolution task, h corresponds to the left-linking tree, which represents, for each mention, one of the previous mentions that belongs to the same cluster. Latent SSVM (36) extends SSVM for training with hidden variables using the Concave-Convex Procedure (CCCP). The key idea is to use the current weights to perform a maximization over hidden

variables (say h^*) and use h^* as the ground truth for training via SSVM training. These two steps are repeated for some fixed number of iterations or until convergence.

4.1.2 Search-based Inference. Our beam search inference procedure consists of the following components: 1) Search space; 2) Beam width b; and 3) Number of search steps. The beam search is guided by a scoring function of the form $w \cdot \Phi(x, y)$ until it reaches a locally optimal state with output $\hat{y}$ (see the supplementary material for pseudo-code).

Search Space. Each state in our search space is a partial (for pipeline approach) or complete (for joint and cyclic approaches) multi-task structured output. The successor function `Succ` generates a successor state by executing an action $a(i, j, z)$, which indicates changing ith slot of parent output y of task j to a new value z, where $z \neq y_i^j$, $z \in C(y_i^j)$, and retaining the values of all other output variables unchanged. For both pipeline and cyclic architectures, we only consider changes to the output variables of a single task until it is finished (task j is fixed). On the other hand, for joint architecture, we consider changes to all output variables of all tasks at every step, which results in a large branching factor.

We design a simple, but effective search space to reduce the depth at which target outputs can be found. To bootstrap the search, we initialize the search with the output obtained from predictions of the i.i.d classifiers $r_1, r_2, \ldots, r_k$ for the k structured prediction tasks learned using only the unary features. The initial search state $I(x)$ is equal to $y_0 = (r_1(x), r_2(x), \ldots, r_k(x))$. Intuitively, we expect that starting from the output of i.i.d classifiers will only need a small number of corrections to reach the target output.

Beam Search Procedure. The beam is initialized with $I(x)$, the output from i.i.d classifiers. Each search step picks the best node in the beam according to the scoring function (selection), generates all its successors by calling `Succ` function (expansion), and updates the beam with the top-b scoring nodes in the candidate set (pruning), where b is the beam width. The search continues until the maximum number of steps or a local optima is reached. Beam search is a tradeoff between greedy ($b = 1$) and pure best-first search ($b = \infty$) and maintains tractability in terms of time and space to produce high scoring outputs.

4.2. Pipeline Architecture

The pipeline architecture requires an ordering over all the k tasks. Suppose Π denotes an ordering over all the tasks, where $\Pi(i)$ denotes the i^{th} task in the order.

Model. We learn one model $\mathcal{M}_i$ (weight vector) to predict the output variables for task $\Pi(i)$ in a sequential manner.

Making Predictions. Given an input x and learned models $(M_1, M_2, \cdots, M_k)$, we predict the multi-task structured output as follows. Run beam search guided by M_1 to predict $\hat{y}_1$. For predicting $\hat{y}_{i+1}$, we use the context of predictions $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_i$, and perform beam search guided by M_{i+1} in the search space of candidate outputs for task $\Pi(i + 1)$.

Learning. We train the models $M_1, M_2, \cdots, M_k$ sequentially as in stacking (6) and forward training. Model M_1 is trained such that for each training input x, the score of the ground truth output y_1^* is higher than all other candidate outputs. We train model M_{i+1} conditioned on the outputs of the learned models $M_1, M_2, \cdots, M_i$ with no sharing of parameters. Specifically, for each training input x, the score of $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_i, y_{i+1}^*$ is higher than the

score of $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_i, y_{i+1}$, where y_{i+1} is any wrong output for $\Pi(i+1)$, and $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_i$ correspond to the predictions of $M_1, M_2, \cdots, M_i$.

The training and inference in the pipeline architecture is very fast. However, it suffers from two drawbacks that can be detrimental to overall accuracy: 1) its sensitivity to the the task ordering Π, and 2) its susceptibility to error propagation.

4.3. Joint Architecture

In the joint architecture, we learn a single model (weight vector) to score a given structured input x and candidate multi-task structured output $y = (y^1, y^2, \cdots, y^k)$ pair. Given an input x and the learned model, we predict the multi-task output by performing beam search in the joint search space. Learning is performed similarly to the single-task structured prediction.

Motivation for Pruning. Our formulation of beam search for MTSP problems in joint architecture suffers from a large branching factor, which is equal to the total number of output variables times the number of candidate values. Specifically, the number of successors or branching factor is $O(kTd)$, where k is the number of tasks, T is the number of output variables, and d is the average size of candidate value set $C(y_i^j)$. For our entity analysis problem, $k = 3$ and T is the number of mentions. The size of the candidate value set $|C(y_i^j)|$ is generally small for entity typing (number of tags) and entity linking (number of candidate entries extracted from KB), but it is very large for coreference resolution (all antecedent mentions: $O(T)$). For example, some large documents typically contain 300 mentions with a branching factor larger than 9000 for all tasks combined. We address this problem by learning pruning functions to create sparse search spaces.

Pruning Mechanism. The pruning method considers all candidate label changing actions $\mathcal{A}(I)$ available at the initial search state and selects the top-scoring $\alpha\,|A(I)|$ actions $\mathcal{A}_p$ using a learned pruning function P, where $\alpha \in [0, 1]$ is a pruning parameter. Only actions from $\mathcal{A}_p$ will be used throughout the search process. This reduces the branching factor from $|A(I)|$ to $\alpha\,|A(I)|$, and can significantly improve the speed of inference for small values of α. The key challenge is to learn an effective pruning function that prunes all but α fraction of the actions, while still retaining near-optimal solutions in its space.

Learning Choices. We can learn pruning function in two different ways. **1) Score-agnostic:** First learn pruning function P to create a sparse search space and then learn a scoring function using the sparse search space created by the learned P. This approach will speed up both training and test-time inference. **2) Score-sensitive:** First learn the scoring function over complete search space and then learn a pruning function to retain or improve the accuracy of search with the learned scoring function. The second approach will only improve the speed of test-time inference. However, it might improve the accuracy over the complete search by pruning nodes where the scoring function is inaccurate.

Pruning Function Learning. We formulate pruning function learning as a rank learning problem. This allows us to leverage powerful off-the-shelf rank learning algorithms. Given the actions $\mathcal{A}(I)$ available at the initial search state, we consider any label changing action $a \in \mathcal{A}(I)$ that improves the accuracy over the initial output as a *good action*; otherwise, it is a *bad action*. We assume the availability of a feature function Ψ over state-action

pairs. We define Ψ as the difference between the features of child state and parent state: $\Psi(a) = \Phi(x, y_{chld}) - \Phi(x, y_{prnt})$. We study the following two pruning approaches.

1. Score-Agnostic Pruning: We represent a bipartite ranking example in the form $(Q > Q')$, which means that in the list $Q \cup Q'$, every element in Q should be ranked higher than every element in Q'. In the first iteration, we collect one bipartite ranking example of actions from each MTSP training example: $(GOOD(I) > BAD(I))$, where $GOOD(I)$ and $BAD(I)$ refer to the set of good and bad actions from $\mathcal{A}(I)$ respectively. The aggregate set of ranking examples $\mathcal{R}$ is given to a rank learner to induce a pruning function P. Then a scoring function is trained in the sparse space defined by $\mathcal{A}_p$.

In subsequent iterations, we collect additional ranking examples based on the mistakes of the current P on each MTSP training example. Define $\mathcal{A}_p^- = \mathcal{A}(I) \setminus \mathcal{A}_p$, and $M = GOOD(I) \cap \mathcal{A}_p^-$. We consider a ranking result of $\mathcal{A}(I)$ as a mistake if $M \neq \emptyset$. Once there is a mistake, we collect a new bipartite ranking example: ($M >$ bottom $|M|$ bad actions in $\mathcal{A}_p$), and then add it to $\mathcal{R}$. The rationale is that we need to get the good actions in M into A_p, and we can do this with minimal pair swapping by pushing the worst-scoring bad actions out of A_p. The pruning function P is re-learned using the updated $\mathcal{R}$.

2. Score-Sensitive Pruning: In this case, we give the weights of the scoring function learned in the complete search space as input to the pruning function learner. As before, the pruning function is learned iteratively. Suppose P_i is the learned pruning function at the end of iteration i. For input example x, we use $\hat{y}_i$ to denote the prediction of search guided by the learned scoring function in the sparse search space created by $\mathcal{A}_{p_{i-1}}$, and $M_i \subseteq \mathcal{A}(I)$ be the set of actions corresponding to mistakes (incorrect outputs) in $\hat{y}_i$. The key idea behind learning is to keep the actions in $\bigcup_{j=1}^{i} M_j$ outside $\mathcal{A}_p$ to improve the accuracy of search with scoring function. We initialize P_0 to a random scoring function. In each iteration i, we initialize ranking example set $\mathcal{R} = \emptyset$. Let $M_i' = \bigcup_{j=1}^{i} M_j$. For each MTSP training example, if $M_i \neq \emptyset$, we create one bipartite ranking example as follows. To the actions in $GOOD(I)$ we add the top few actions in $BAD(I) \setminus M_i'$ according to P_{i-1} to get a set $GOOD'$ which is no larger than $\mathcal{A}_{p_{i-1}}$. We then add the example $GOOD' > M_i'$ to $\mathcal{R}$. The pruning function P_i is learned using ranking examples $\mathcal{R}$ at the end of iteration i.

4.4. Cyclic Architecture

The cyclic architecture lies in between pipeline and joint architectures in terms of the overall complexity. It retains the efficiency of pipeline architecture by focusing on one task at a time, but tries to avoid its weaknesses of dependence on task order and error propagation by cycling through the tasks mutiple times.

Model. We learn one model $\mathcal{M}_i$ (weight vector) to predict the output variables for task $\Pi(i)$ in a sequential manner.

Making Predictions. Given an input x and learned model $(M_1, M_2, \cdots, M_k)$, we predict the multi-task structured output as follows. We initialize the output of all k tasks using i.i.d classifiers (say $y(0)$). We perform sequential inference using the models until convergence (multi-task output does not change in two consecutive cycles) or for maximum number of cycles. In each cycle, we make predictions for tasks in the same order as Π via beam search using the most recent predictions for all other tasks as context.

Learning. We train the models $M_1, M_2, \cdots, M_k$ sequentially as in the pipeline architecture with two differences: 1) The model for each task is trained conditioned on the most recent model (or i.i.d classifier) for all the other tasks; and 2) The training process is repeated for a fixed number of cycles to further improve the models. The number of training cycles are determined based on the performance on development data.

We explored two different versions of the cyclic architectures. In one, there is a single set of shared weights for all tasks. In the second, each task has its own set of weights even for shared inter-task features. While sharing weights could improve statistical efficiency, it also results in reduced expressiveness which is sometimes detrimental to accuracy.

5. Experiments and Results

We evaluate our approach on two annotated datasets, ACE 2005 corpus (25) and TAC-KBP 2015 Entity Linking corpus (17), on three entity analysis tasks: named entity recognition, coreference resolution, and entity linking. For both corpora, we first report the results of learning and inference using the complete search spaces, and show that we can achieve comparable or better performance than state-of-the-art approaches. Next, we present the results for pruned and cyclic architectures to show the improvements in speed and accuracy.

5.1. Experimental Setup

Datasets. *ACE 2005* corpus contains 599 English documents. We follow the same setting as (13) to make a train/dev/test split of 338/144/117, so that all the results are comparable. The original ACE 2005 has included the gold annotation for coreference and NE types, but does not contain the entity linking annotation. In order to do entity linking model training and evaluation, we add ACE-to-Wiki annotation (1) to the corpus. Therefore, the linking task on ACE 2005 will use Wikipedia as the KB. *TAC-KBP 2015 Entity Linking* corpus is released for TAC-KBP 2015 Tri-lingual Entity Discovery and Linking (TEDL) task (17).

Evaluation Metrics. The ACE 2005 evaluation follows the standard metrics for each task. Coreference results are evaluated using MUC, B^3, $CEAF_e$, and the average of these three metrics called the CoNLL metric (26). We employed the official CoNLL scorer to compute scores. For NER typing, results are scored using Hamming accuracy. We evaluate the entity linking result on overall accuracy, which is just the percentage of mentions that are linked correctly. Note that for entity linking in ACE 2005, only the proper and nominal mentions will be considered because ACE-to-Wiki annotation does not include pronouns.

Following the TAC-KBP EL evaluation procedure (17), the metrics for scoring typing and linking are the same as ACE 2005. Additionally, the competition also computes a score called *NERLC*, where a decision is scored correct if both the mention's type and its linked KB ID are correct. The clustering is evaluated by cross document $CEAF_m$ (23). We also report the within document coreference scores using the same metrics as ACE 2005 (25). Note that the TAC competition not only requires the system to link each mention to its correct KB ID entry when it exists, but also to cluster the NIL mentions (mentions without links) according to their coreferent similarities. We employ a simple rule-based agglomerative clustering approach similar to the Stanford multi-sieve system to perform NIL clustering (21). All the reported scores are computed through the official scoring script. For all the three tasks, we assume that the gold mention boundaries are given.

System Implementation Details. Our entity analysis system is developed based on Berkeley-Entity-Resolution system (13). We replace the learning and inference components

with our search-based structured prediction approach. For the experiments on ACE 2005, documents are preprocessed using OpenNLP-tokenizer and Berkeley-Parser. For TAC-KBP 2015 corpus, we employ StanfordCoreNLP pipeline to do all the pre-processing.

We employ `Illinois-SL` (3) structured learning library for latent SSVM training with maximum number of CCCP iterations set to 10. The scoring function is trained to optimize the Hamming loss. For named entity typing and entity linking, the Hamming loss is simply the classification error over all mentions, while for coreference, we compute the Hamming loss using the proportion of mentions with the wrong left-linked antecedent (i.e. inconsistent with the ground truth clustering). For pruning function learning, we employed `XGBoost` (4) library to learn boosted regression trees with pairwise ranking loss as our training objective. The learning algorithm for pruning is run for 5 iterations. All hyper-parameters are tuned using the development data.

Before running our system for entity linking, two prerequisite models need to be prepared. First is the scored lookup table $\mu : q \rightarrow E$ for candidate generation. Our system assumes that the candidate KB entries of mentions are generated before doing the joint learning or inference. Our candidate generation system takes a mention span s_m as input, generates a query set $Q(s_m)$ by taking its substrings or expansions, probes μ for each $q \in Q$ to get scored candidate entity set E_q, and finally, returns top-L scored candidates among $\bigcup_Q E_q$. The score of each query-entity pair, denoted by $g(q, e)$, is the weighted sum of number of times the map $q \rightarrow e$ appeared in titles and hyperlinks in the entire KB. Besides candidate generation, our TAC linking feature also requires mention and entity embeddings for computing similarity. Similar to the Alignment by Anchor model of (33), we learn word, mention and entity embeddings by applying the skip-gram model (24) to the training set created from the English Wikipdia corpus after modifying them by adding the anchor text and anchor entity to the words in the sentences.

Features. Recall that the joint feature vector in Equation 1 consists of the intra-task and inter-task features. This can be further decomposed as follows:

$$\begin{aligned}\Phi(x,y) \quad = \quad & \Phi_c(x,y_c) \circ \Phi_n(x,y_n) \circ \Phi_l(x,y_l) \circ \sum_i \phi_{(c,n)}(m_i, m_j, y_i^n, y_j^n) \circ \\ & \sum_i \phi_{(c,l)}(m_i, m_j, y_i^l, y_j^l) \circ \sum_i \phi_{(n,l)}(m_i, y_i^n, y_i^l), \text{ where } j = y_i^c \end{aligned} \tag{2}$$

where $\phi_{(t,t')}$ is the inter-task feature extracted from a mention or a coreferent mention pair and its corresponding predictions of tasks t and t'. The sums are vector sums over all mentions m_i, and since all mention pair interactions are confined to left links, $j = y_i^c$.

For ACE 2005 corpus, we follow the same feature design as the Berkeley system (13) for both intra and inter task features. We define $\phi_c(m', m)$ as the intra-coreference features over mention pair, and $\phi_n(m, \tau)$ and $\phi_l(m, e)$ as unary intra NER and linking features between mention m and its corresponding tag and KB entry. Since we treat a query-entry pair (q, e) as one value in our formulation, $\phi_l(m_i, e)$ is just the concatenation of the feature vector over (m_i, q) and the vector over (q, e) in the Berkeley system.

For TAC-KBP 2015 corpus, we employ the same features as ACE 2005 for ϕ_c, ϕ_n, and $\phi_{(n,c)}$. For linking, we drop the query variable, and instead use a learned embedding space to compute the cosine similarity between a mention and a KB entry, and employ this distance as one of the features. We re-design features ϕ_l, $\phi_{(c,l)}$, and $\phi_{(n,l)}$ as follows:

- $\phi_l(m_i, e)$ includes `CandidateGenScore` and `CosineSimilarity` which represent similarity scores between m_i and e computed using a heuristic function or through the mention-entity embedding; `ExactMatch, SubString`, and `SameInitial` which capture surface similarities; and `HashDescrip, HasType`, and `HasWebsites` that indicate how informative e is;
- $\phi_{(c,l)}(m_i, m_j, y_i^l, y_j^l)$ includes `SameKBID, SameFreebaseType`, and `SharedRelatedWebsite` which use the properties of linked KB entry of each mention to measure the coreference consistency between m_i and m_j.
- $\phi_{(n,l)}(m_i, y_i^n, y_i^l)$ includes `NerFreebaseTypePair` and `NerFreebaseParentTypePair`, which model a weighted soft map from the assigned Freebase type and its parent type in Freebase type system of mention m_i to the NER types.

Hyper-parameters. In scoring function learning, the C parameter in the latent SSVM model was tuned using the average hamming accuracy over all tasks on the development set, and was set to be 0.0001 for both corpora, and fixed in all the experiments. In pruning function learning with `XGBoost`, we set the maximum tree depth to be 20, and maximum boosting iterations to be 500. The pruning parameter α was selected based on the performance on development set.

5.2. Beam Size Analysis

We considered candidate beam widths b from $\{1, 5, 10, 20, 40, 60\}$. We performed experiments over development set of the two datasets in the complete search space with different b values. Results in Figure 1 show that larger beam size is useful in overcoming the local optima challenge, but improvement becomes small when b is larger than 20. We conservatively fixed $b = 40$ for all our experiments and other hyper-parameters are tuned accordingly.

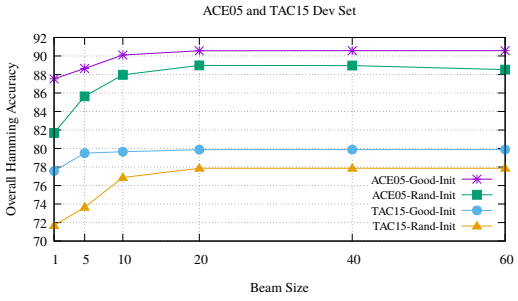

Figure 1: ACE 2005 and TAC-KBP 2015 Dev Set Performance with different beam sizes.

5.3. Results for Single-Task Structured Prediction and Pipeline Architecture

One simple approach to handle multi-task structured prediction problems is to perform stacked training and inference, where the output of one task is fed as input to provide context for solving the next task in the pipeline (6; 28). However, the pipeline approach requires an ordering of the tasks, which may be hard to fix without significant domain knowledge. Therefore, we considered all possible orderings over the tasks (6 for 3 tasks) in our experiments. We refer to the three tasks as CR, NER, and EL. All the results above will also be compared with the single-task structured prediction (STSP) approach, where each task is solved independently.

Table 1 shows the results of the pipeline approach with different task orderings and the STSP approach. We can make two observations. First, the performance of the tasks is better when they are placed later in the ordering. It is especially true for NER and EL tasks. This shows that dependencies between tasks exist and can be leveraged to improve the performance. Second, there is no ordering that allows the pipeline approach to reach peak

Algorithms	ACE 2005 Test			TAC-KBP 2015 Test			
	Coref.	NER	Link	NER	Link	NERLC	w.in Coref.
STSP	75.04	82.24	75.35	87.30	76.20	70.90	81.21
CL→NER→EL	75.04	83.41	77.15	87.90	76.10	71.22	81.21
CL→EL→NER	75.04	85.27	74.64	86.80	75.70	70.71	81.21
NER→CL→EL	76.20	82.24	77.23	87.30	76.50	71.33	82.47
NER→EL→CL	76.50	82.24	76.77	87.30	74.90	69.96	82.62
EL→NER→CL	76.66	84.77	75.35	87.10	76.20	71.01	81.38
EL→CL→NER	76.08	85.08	75.35	88.20	76.20	71.89	79.89

Table 1: ACE 2005 and TAC 2015 Test Set Performance with different pipeline orderings.

performance on all the three tasks. This is due to the inherent limitations of the pipeline approach: mistakes in earlier tasks can hurt the performance of downstream tasks, and the architecture does not allow to revisit/correct predictions based on additional evidence(s). These observations corroborate that a more global learning/inference approach may do better than both pipeline and STSP approaches.

5.4. Results for Joint Architecture without Pruning

In this section, we report the results of our entity analysis system with beam search in the complete search space. Tests using the paired bootstrap resampling approach indicate that the performance differences we observe are statistically significant in all three tasks.

Algorithm	Coreference				NER	Link	Train
	MUC	BCube	$CEAF_e$	CoNLL	Accu.	Accu.	time
Berkeley	81.41	74.70	72.93	76.35	85.60	76.78	31min
a. Results of Joint Architecture without Pruning							
STSP	80.28	73.26	71.58	75.04	82.24	75.36	9min
Joint w. Rand Init	80.23	73.79	72.03	75.35	82.20	76.99	48min
Joint w. Good init	82.18	**76.57**	74.00	77.58	85.71	78.77	34min
b. Results of Joint Architecture with Pruning							
Score-agnostic	81.10	75.79	74.33	77.07	85.63	78.71	16min
Score-sensitive	**82.81**	75.77	**74.96**	**77.85**	**87.18**	80.28	37min
c. Results of Cyclic Architecture							
Unshrd-Wt-Cyclic	81.83	76.05	73.99	77.29	84.18	**80.67**	11min
Shared-Wt-Cyclic	80.97	75.22	73.39	76.53	82.16	79.60	10min

Table 2: ACE 2005 Test Set Performance.

Table 2.a shows the performance on ACE 2005 testing set for all 3 tasks. `Berkeley` (13) is our baseline result. `STSP` is the result without using inter-task features. `Joint-Rand-Init` and `Joint-Good-Init` are the results of joint search-based architecture with random initial state and the output of `STSP` respectively. We can draw three conclusions from this table. First, the difference between `STSP` and `Joint-Good-Init` shows that exploiting the interdependency between the tasks, which is captured by inter-task features, does benefit the system performance on all tasks. Second, we can see that `Joint-Good-Init` significantly outperforms `Joint-Rand-Init`, which shows that search-based inference for large structured prediction problems suffers from local optima and is mitigated by a good initialization. Finally, our search-based MTSP predictor is competitive or better than the state-of-the-art system for entity analysis.

2. corresponds to `NERC` metric in the official report.

Algm.	NER[2] Accu.	Link Accu.	NERLC Accu.	Within. Coref MUC	BCub	$CEAF_e$	CoNLL	Crs.Crf $CEAF_m$	Trn. time
Rank-1st	87.0	-	73.7	-	-	-	-	80.0	-
Berkeley	88.90	74.80	72.80	86.02	83.66	79.27	82.98	80.8	6m29s
a. Results of Joint Architecture without Pruning									
STSP	87.30	76.20	70.90	84.29	82.04	77.30	81.21	78.8	2m41s
Joint Rnd. Ini	87.10	71.17	68.33	84.34	82.14	77.45	81.31	78.4	7m19s
Joint Gd. Ini	**89.72**	76.98	74.43	85.87	83.48	79.05	82.80	81.3	6m11s
b. Results of Joint Architecture with Pruning									
Score-agnostic	89.54	76.84	74.31	85.57	84.04	**79.38**	82.99	**81.4**	4m15s
Score-sensitive	89.33	**77.68**	**74.63**	**86.08**	**84.20**	79.22	**83.17**	81.3	9m2s
c. Results of Cyclic Architecture									
Ushrd-Wt-Cyc	89.57	77.68	74.60	84.97	83.08	78.18	82.08	80.5	3m52s
Shard-Wt-Cyc	87.95	73.65	71.32	82.67	81.09	77.86	80.54	77.9	2m56s

Table 3: TAC-KBP 2015 EL Test Set Performance.

Table 3.a presents our results on TAC-KBP 2015. In this table, we add one more baseline `Rank-1st` (17), which is the best performing system in TAC-KBP 2015 EL competition. The NERLC is the official joint metric of entity linking and NER typing performance, and $CEAF_m$ is the official metric for clustering (17). Again, our joint system outperforms both baselines by at least 1%, and the results show the same trend as ACE 2005. If we compare the difference between the Link and NERLC, we can see that, `STSP` loses 6% accuracy from Link to NERLC, while both joint runs only drop less than 2% accuracy. These differences show the importance of the joint architecture and the inter-task features.

5.5. Results for Joint Architecture with Pruning

In this section, we report the results of our entity analysis system with beam search in the pruned search space.

Table 2.b and 3.b show a comparison of test results with and without pruning for both corpora. `Score-Agnostic` corresponds to the result of learning the pruning function before learning the scoring function. `Score-Sensitive` is the result of learning the pruning function based on the scoring function. By comparing these tables with Table 3, we can see that `Score-Agnostic` achieved a competitive performance with `Joint-Good-Init` in about half the training time. Furthermore, `Score-Sensitive` has outperformed `Joint-Good-Init`, which shows that a score-sensitive pruner could correct mistakes of the scoring function, and bring potential accuracy improvements.

Score-agnostic Pruning. Table 4 is a study of how the accuracy and training time would change with respect to the pruning parameter when a pruner is learned before the scoring function. As the table shows, when α becomes larger, the development set performance gradually recovers to the level of no-pruning performance, while the training time increases gradually. The performance loss with small α is mainly caused by the recall loss during the pruning.

Score-sensitive Pruning. Table 5 shows the results of applying the pruner learned with different α's after the cost function was learned on the development sets. As α increases, the performance of the three tasks goes up to a maximum, and then slowly goes down. On ACE 2005 and TAC 2015, these optimal points are reached at $\alpha = 0.6$ and $\alpha = 0.7$ respectively. By carefully adjusting α, the pruner would become tuned to the scoring function, and improves performance.

Prn. α	ACE05 Dev Hamm.	ACE05 Dev Trn.Tm	TAC15 Dev Hamm.	TAC15 Dev Trn.Tm
0.3	86.53	11m	75.21	2m11s
0.5	**90.83**	16m	77.27	3m08s
0.7	90.80	22m	**80.60**	4m15s
0.9	90.67	29m	80.40	5m44s
1	90.58	34m	80.47	6m11s

Table 4: ACE 2005 and TAC 2015 Dev Set accuracy w.r.t. α with score-agnostic pruning function. α starts from 0.3 because this pruning learning requires $\alpha|\mathcal{A}(I)| > |GOOD(I)|$ on average over training set.

Prn. α	ACE05 Dev Hamm.	ACE05 Dev Tst.Tm	TAC15 Dev Hamm.	TAC15 Dev Tst.Tm
0.1	81.61	45s	75.61	31s
0.3	87.29	1m17s	78.33	48s
0.5	89.15	1m58s	79.79	1m02s
0.6	**90.80**	2m10s	80.51	1m15s
0.7	90.78	2m17s	**80.83**	1m41s
0.9	90.62	2m25s	80.66	1m55s
1	90.58	2m44s	80.47	2m05s

Table 5: ACE 2005 and TAC 2015 Dev Set accuracy w.r.t. α with score-sensitive pruning.

5.6. Results for Cyclic Architecture

Table 2.c and 3.c show a comparison of test results of the two cyclic training approaches on both corpora. `Unshared-Wt-Cyclic` and `Shared-Wt-Cyclic` correspond to the cyclic training method with 3 different task-specific weight vectors and with one single weight vector, respectively. We tune the parameters (training cycles and task ordering) according to the overall hamming accuracy on development set.

`Unshared-Wt-Cyclic` can reach a comparable performance to `Joint-Good-Init` on coreference and linking, but is slightly weaker on NER. `Shared-Wt-Cyclic` performs worse than `Unshared-Wt-Cyclic`, especially on NER. Importantly, both cyclic architectures have a big advantage in training time, even compared to the joint architecture with pruning. In each task of each cycle, they only perform training and inference on a single task, which is of similar time complexity to STSP. The search in STSP is faster than joint architecture due to reduced branching factor as well as search depth.

As can be seen in Table 2, `Shared-Wt-Cyclic` does not perform as well as `Unshared-Wt-Cyclic`. This is because when the different tasks share one weight vector, the inter-task features of the weights are updated in two different task stages in each cycle. When the optimal weights for each task are slightly different from the other task, the latter task overwrites the former task's learned weights, and vice versa, thus undermining each other. As a result, only the last task in the ordering can fully exploit the inter-task features.

	ACE 2005 Dev Set					
Cycle task orderings	Unshared-Wt-Cyclic Coref.	NER	Link	Shared-Wt-Cyclic Coref.	NER	Link
CL→NER→EL	76.63	84.24	80.06	72.01	82.94	77.30
CL→EL→NER	76.66	84.54	80.01	73.20	84.87	76.17
NER→CL→EL	77.14	84.32	79.93	75.89	83.45	77.29
NER→EL→CL	77.20	84.14	80.00	74.95	83.03	72.39
EL→NER→CL	76.64	84.16	80.21	75.24	84.18	75.21
EL→CL→NER	77.19	84.32	80.08	76.08	84.16	73.43

Figure 2: Unshared-Wt-Cyclic and Shared-Wt-Cyclic performance on ACE 2005 Dev w.r.t. task orderings.

To verify our hypothesis, we present the ACE05 dev scores of the two algorithms in table 2. Each row corresponds to one task ordering. It is easy to observe that compared to `Unshared-Wt-Cyclic`, in `Shared-Wt-Cyclic` columns only the last tasks perform relatively well, while the first task can only reach a score slightly better than the initialization.

In our cyclic approach, there is no need to restrict the testing cycle number to be exactly the same with the the training cycle number. To determine the proper number of testing cycle number, we did a study in which we plotted the testing accuracy on dev set with

different test cycles using the current cyclic model. Our study shows that for both datasets, and in all task orderings, the best accuracy can be reached after 3 to 4 cycles, and is stable afterward. In testing, since there is no disadvantage for increasing the number of cycles, we can do as many cycles as there is time for. We stopped the cycles once more than 95% of the predicted outputs did not change in the last two cycles.

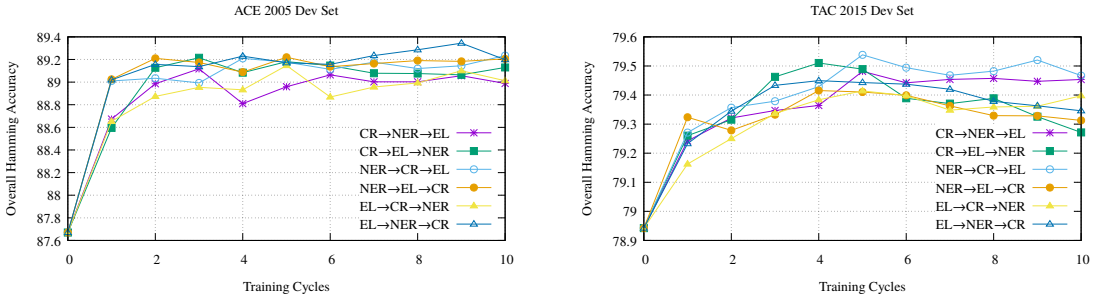

Figure 3: Hamming accuracy on ACE05 (l) and TAC15 (r) Dev w.r.t. Unshred-Wt-Cyc training cycles.

In Figure 3 we present the overall hamming accuracy w.r.t. the number of training cycles for `Unshared-Wt-Cyclic`. The accuracy at the 0^{th} cycle is computed from initial outputs. The figure shows that, for both datasets, our cyclic training approach can continuously improve the accuracy in the first 2 to 4 cycles, regardless of task ordering. However, unlike during the testing phase, too many training cycles could lead to overfitting. We selected the best number of training cycles using the development set performance.

6. Summary

We studied the problem of multi-task structured prediction (MTSP) in the context of entity analysis of natural language text. We developed a search-based learning framework, where we employed structured SVM for training and beam search for inference. To improve the efficiency of training and test-time inference, we learned pruning functions to create sparse search spaces. Our joint search architecture improves on both accuracy and speed over the state-of-the-art graphical modeling approach. We also explored a cyclic architecture which is highly efficient and is competitive with the joint search.

Acknowledgements. This work is supported in part by the grants from DARPA, including XAI (N66001-17-2-4030) and DEFT (FA8750-13-2-0033); and from NSF, including CNS-1543656, IIS-1619433, and IIS-1219258. The first author would like to thank Greg Durrett, Shyam Upadhyay, and Kai-Wei Chang for their help with software systems.

References

[1] Luisa Bentivogli, Pamela Forner, Claudio Giuliano, Alessandro Marchetti, Emanuele Pianta, and Kateryna Tymoshenko. Extending English ACE 2005 Corpus Annotation with Ground-truth Links to Wikipedia. In *TPWMNLP Workshop*, 2010.

[2] Bernd Bohnet and Joakim Nivre. A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing. In *EMNLP-CoNLL*, 2012.

[3] Kai-Wei Chang, Shyam Upadhyay, Ming-Wei Chang, Vivek Srikumar, and Dan Roth. Illinois-SL: A JAVA Library for Structured Prediction. In *arXiv:1509.07179*, 2015.

[4] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *KDD*, 2016.

[5] Xiao Cheng and Dan Roth. Relational Inference for Wikification. In *EMNLP*, 2013.

[6] William W. Cohen and Vitor Rocha de Carvalho. Stacked Sequential Learning. In *IJCAI*, 2005.

[7] Hal Daumé, III and Daniel Marcu. A Large-scale Exploration of Effective Global Features for a Joint Entity Detection and Tracking Model. In *HLT-EMNLP*, 2005.
[8] Hal Daumé, III and Daniel Marcu. Learning As Search Optimization: Approximate Large Margin Methods for Structured Prediction. In *ICML*, 2005.
[9] Hal Daumé, III, John Langford, and Daniel Marcu. Search-based Structured Prediction. *MLJ*, 2009.
[10] Pascal Denis and Jason Baldridge. Joint Determination of Anaphoricity and Coreference Resolution using Integer Programming. In *HLT-NAACL*, 2007.
[11] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A Learning Framework for Search-based Structured Prediction. *JAIR*, 49, 2014.
[12] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured Prediction via Output Space Search. *JMLR*, 15, 2014.
[13] Greg Durrett and Dan Klein. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. In *TACL*, 2014.
[14] Jenny Rose Finkel and Christopher D. Manning. Joint Parsing and Named Entity Recognition. In *NAACL*, 2009.
[15] Hannaneh Hajishirzi, Leila Zilles, Daniel S. Weld, and Luke S. Zettlemoyer. Joint Coreference Resolution and Named-Entity Linking with Multi-Pass Sieves. In *EMNLP*, 2013.
[16] Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *ACL*, 2012.
[17] Heng Ji, Joel Nothman, Ben Hachey, and Radu Florian. Overview of TAC-KBP2015 Tri-lingual Entity Discovery and Linking, 2015.
[18] Jonathan K. Kummerfeld, Taylor Berg-Kirkpatrick, and Dan Klein. An Empirical Analysis of Optimization for Max-Margin NLP. In *EMNLP*, 2015.
[19] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*, 2001.
[20] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Tom Dietterich. HC-Search for Structured Prediction in Computer Vision. In *CVPR*, 2015.
[21] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's Multi-pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In *CoNLL: Shared Task*, 2011.
[22] Qi Li and Heng Ji. Incremental Joint Extraction of Entity Mentions and Relations. In *ACL*, 2014.
[23] Xiaoqiang Luo. On Coreference Resolution Performance Metrics. In *HLT-EMNLP*, 2005.
[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, 2013.
[25] NIST. The ACE Evaluation Plan, 2005.
[26] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Modeling Multilingual Unrestricted Coreference in OntoNotes. In *EMNLP-CoNLL: Shared Task*, 2012.
[27] Lev Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *CoNLL*, 2009.
[28] Stéphane Ross and Drew Bagnell. Efficient Reductions for Imitation Learning. In *AISTATS*, 2010.
[29] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *JMLR*, 15, 2011.
[30] Dan Roth and Wen-tau Yih. Integer Linear Programming Inference for Conditional Random Fields. In *ICML*, 2005.
[31] Sameer Singh, Sebastian Riedel, Brian Martin, Jiaping Zheng, and Andrew McCallum. Joint Inference of Entities, Relations, and Coreference. In *AKBC Workshop*, 2013.
[32] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *ICML*, 2004.
[33] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph and Text Jointly Embedding. In *EMNLP*, 2014.
[34] David Weiss and Ben Taskar. Structured Prediction Cascades. In *AISTATS*, 2010.
[35] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning Linear Ranking Functions for Beam Search with Application to Planning. *JMLR*, 2009.
[36] Chun-Nam John Yu and Thorsten Joachims. Learning Structural SVMs with Latent Variables. In *ICML*, 2009.