AN ABSTRACT OF THE THESIS OF

Chao Ma for the degree of Doctor of Philosophy in Computer Science presented on

July 15, 2019.

Title: New Directions in Search-based Structured Prediction: Multi-Task Learning and Integration of Deep Models

Abstract approved: ____

Prasad Tadepalli   Janardhan Rao Doppa

This thesis studies the problem of structured prediction (SP), where the agent needs to predict a structured output for a given structured input (e.g., Part-of-Speech tagging sequence for an input sentence). Many important applications including machine translation in natural language processing (NLP) and image interpretation in computer vision can be naturally formulated as structured prediction problems. These prediction problems have an exponentially number of candidate outputs, which poses significant challenges for inference and learning. Search-based structured prediction views the prediction of structured outputs as a search process in the space of candidate outputs guided by a learned scoring function. Search-based approaches offer several advantages including, incorporation of expressive representations over inputs and outputs with negligible overhead in inference complexity, and providing a way to trade off accuracy for efficient inference. In this thesis, I make three contributions to advance search-based structured prediction methods towards the goal of improving their accuracy and computational-efficiency.

First, I developed a search-based learning approach called "*Prune-and-Score*" to improve the accuracy of greedy policy based structured prediction for search spaces with large action spaces. The key idea is to learn a pruning function that prunes bad decisions and a scoring function that then selects the best among the remaining decisions. I show the efficacy of this approach for coreference resolution, i.e., clustering mentions that refer to the same entity, which is a hard NLP task.

Second, I studied multi-task structured prediction (MTSP) problems in the context of entity analysis, which includes the three tasks of named entity recognition, co-reference resolution, and entity linking. I developed three different search-based learning architectures for MTSP problem that make different trade-offs between speed and accuracy of training and inference. I performed empirical evaluation of the proposed architectures for entity analysis with state-of-the-art results.

Finally, I developed the *HC-Nets* framework by integrating the advances in deep learning with search-based SP methods. I formulate structured prediction as a complete output space search problem, and employ two neural network functions to guide the search: a heuristic function $\mathcal{H}$ for generating candidate outputs and a cost function $\mathcal{C}$ for selecting the best output from the generated candidates. Unlike prior methods such as structured prediction energy networks and deep value networks that perform gradient-based inference in relaxed space, $\mathcal{HC}$-Nets allows to incorporate prior knowledge in the form of constraints. The decomposition of heuristic and cost functions makes the representation more expressive and learning more modular. Our experimental results show that $\mathcal{HC}$-Nets achieves better performance than prior methods on multiple benchmark domains.

New Directions in Search-based Structured Prediction: Multi-Task Learning and Integration of Deep Models

by

Chao Ma

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 15, 2019
Commencement June 2020

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Professor Prasad Tadepalli. As a student without much talent in computer science, I am fortunate that I could have Prof. Tadepalli to be my advisor, who gives enough guidance, patience, trust, and freedom, and allows me to learn and pursue my research ideas over years. He is also a considerate advisor who can always think on the perspective of students. I will never be able to finish this thesis and all related research without the help from him.

I would also thank my co-advisor, Professor Janardhan Doppa. Prof. Doppa is not only my co-advisor, but also my academic brother. We had started to cooperate on research projects when he was in OSU. As a young faculty, Prof. Doppa was willing to spend time discussing about every details about our research, helping me on fundamental knowledge, and encouraging me to persist when coming across hard problems, which helps me to approach the research goal. My research path was started from his inspiration, and I do learn so much from him.

Besides my advisors, I also want to show my appreciation to all professors, teachers and mentors who share their expertise and experiences, and for providing valuable feedback throughout my PhD life. Professor Xiaoli Fern, and Professor Tom Dietterich are the professors who collaborate with my advisor on Deep Language Understanding project, which I was also involved as a student. Prof. Fern and Prof. Dietterich provide a lot of valuable suggestions and ideas to my research work during the project group meetings, and the later NLP group meetings. I would sincerely appreciate to all their selfless help. I am also grateful to have Angelo Restificar and Jun Yu as my mentors during my two summer internships at eBay traffic science team. Without their leading and guidance, I would not be able to accumulate my first piece of experience from industry. In particular, I would like to express my gratitude to the help from Jun Yu. By working with Jun Yu. I learned how to balance the exploration and exploitation when doing research in industry, how to organize all the material in the development in a professional manner. His earnest attitude and diligence towards his work will always be the model to me in my future career life.

On personal aspect, I would like to thank to my families for their support in the past years. First, I would specially thank my wife, Xiangyu Wang, who stayed with me during my entire graduate school journey. Thank to all of her understanding about countless evenings and weekends that I am busy working on my research in the lab. I am fortunate that I can share all the happiness with her in my life. I would also thank to my parents. Without their encouragement and support, I will never be able to reach the place that I am staying at today.

Last but not the least, I want to thank all the friends who once helped me. My life at OSU became colorful and enjoyable because of you.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

Many important problems in artificial intelligence including machine translation in natural language processing (NLP) and image interpretation in computer vision can be naturally formulated as structured prediction problems. These problems require the prediction of multiple labels simultaneously. Given an input vector x, we need to learn a model to predict an output vector y. This problem is called *structured prediction*. Generally, such a model relies on a scoring function $s(x, y)$ to determine how good y is as the output of x. To find the best y with respect to x given $s(x, y)$, the structured prediction model needs the inference algorithm to compute $\mathrm{argmax}_{y \in \mathcal{Y}}\, s(x, y)$. The solution space $\mathcal{Y}$ usually contains an exponential number of candidate outputs, which poses significant challenges for inference and learning.

Structured prediction can achieve better performance than predicting each label individually because it could exploit the dependencies in the structural output. For example, in handwriting recognition, taking the constraints of vocabulary and grammar into account would be helpful for predicting the adjacent letters and words correctly. Similarly, in semantic segmentation, the contiguity and spatial relationships among the pixel or super-pixels may be important to improve the prediction accuracy (e.g., sky should appear above the grass). It is also difficult because with multiple labels, the solution space is exponentially large, therefore a brute-force algorithm that predicts each label sequence will not be scalable. In practice, we hope the prediction can be done not only accurately but also quickly.

Search-based structured prediction approaches are an important class of frameworks that have the potential to address the challenges above. In order to do the structured prediction, a search-based algorithm first needs a definition of search space. The search space specifies the search states, which include pairs of structured inputs and outputs, the action space and the successor functions. Second, a search algorithm is required to specify how the search is conducted. Finally, we need a learning algorithm and a scoring function model. The learned scoring function guides the search algorithm to find the best output in the search space.

Compared with other methods, search-based structured prediction has at least two attractive properties. First, given the formulation of search space, the search algorithm only interacts with the search states using actions. Thus, unlike other approaches, e.g., graphical models, the inference time complexity of search-based algorithms is not sensitive to the complexity of the features of the search states (factor size). Second, the search-based approaches can optimize any arbitrary loss functions since it is only used as a black-box to train the scoring function.

In our first work (Ma *et al.*, 2014), we study the partial output space greedy search, and apply this formulation to coreference resolution problem, an application in NLP domain. Coreference resolution is an NLP task of clustering a set of mentions such that all mentions in the same cluster refer to the same entity. In our formulation, mentions are ordered and decisions are made sequentially. A partial clustering result is a state, and an action either merges an unlabeled mention into a cluster in the current state, or creates a new cluster. The learned model is used as a heuristic for choosing the best action and guides the search at each step. To do this, we motivate and introduce our *Prune-and-Score* approach, in which we learn two distinct functions that make decisions in two steps: a pruning function that prunes bad actions from further consideration and a scoring function that selects the best among the remaining actions. We identify a decomposition of the overall loss of the Prune-and-Score approach into the pruning loss and the scoring loss, and reduce the problem of learning these two functions to rank learning, which allows us to leverage powerful and efficient off-the-shelf rank learners. Some evaluation results on OntoNotes dataset are presented, and show that it compares favorably to several state-of-the-art approaches as well as a greedy search-based approach that uses a single scoring function.

Our second work (Ma *et al.*, 2017) tries to address the structured prediction problem that involves multiple tasks, named *multi-task structured prediction* (MTSP), under a specific application in NLP: entity analysis. Instead of partial output space, we employ complete output space best first beam search as the inference method, and learn linear scoring functions with structural SVM. To investigate the ordering of training and inference over different tasks, we study three different search architectures to solve the MTSP problem, which make different trade-offs between speed and accuracy of training and inference. One is a "*pipeline*" architecture, where the different tasks are solved one after another in sequence. While it has the advantages of simplicity and reduced search space, 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 a severe challenge that its branching factor increases in proportion to the number of tasks, making the search too expensive. We address this problem by learning a pruning function that prunes bad candidate solutions from the search space for better efficiency. Finally, we introduce a third search architecture referred as "*cyclic*", whose complexity is intermediate between the two. 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.

Previous work has shown the effectiveness of complete output space search in structured prediction (Doppa *et al.*, 2012, 2013a, 2014b). The recent success of deep learning also proves the power of deep neural network models (Belanger and McCallum, 2016; Belanger *et al.*, 2017; Gygli *et al.*, 2017; Tu and Gimpel, 2018). Our last piece of work seeks to exploit the advantages of both models and proposes a new search-based framework which extends the earlier work on $\mathcal{HC}$-Search Doppa *et al.* (2012) to neural networks. In particular our approach called $\mathcal{HC}$-nets, learns two real valued functions over input-output sequences: a heuristic function to guide the search for a good solutions in the complete output space and a cost function to rank the solutions to pick the best. Both functions are represented as neural networks and are trained offline from supervised data in the imitation learning framework. Several example applications and preliminary results are presented in the end.

In summary, the thesis makes the following contributions:

- We developed a search-based learning approach called *Prune-and-Score* to improve the accuracy of greedy policy based structured prediction for search spaces with large action spaces. We showed the efficacy of this approach for coreference resolution, a hard NLP task.
- We studied three different search-based learning architectures for multi-task structured prediction problems and evaluated them in the context of entity analysis, a natural language understanding task.
- We proposed a new framework that synergistically combines the advantages of output space search based structured prediction methods and representation learning approaches. The proposed framework is evaluated on multiple benchmark structured prediction tasks with promising results.

Chapter 2: Related Work on Structured Prediction

In this section, we will give an overview of the structure prediction problem and different approaches for solving the problem.

2.1 Structured Prediction Problem

A structured prediction problem specifies a space of structured inputs $\mathcal{X}$, a space of structured outputs $\mathcal{Y}$, and a non-negative output loss function $l : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. Each structured input $x \in X$ is presented by a continuous or discrete vector. We use $\hat{y}$ to denote the predicted output and y^* the true output for a given input. $l(x, \hat{y}, y^*)$ is the loss associated with labeling a particular input x by output $\hat{y}$ when the true output is y^* (Deshwal *et al.*, 2019). Note that this function is different from the loss used as the learning objective in training. When $\hat{y}$ and y^* are exactly the same, l would be 0 because there is no error in prediction. Otherwise the larger the l, the more erroneous $\hat{y}$ is. Without loss of generality, we assume that each structured output $y \in Y$ is represented using T discrete and/or continuous variables v_1, v_2, $\cdots$, v_T, and each variable v_i can take candidate values from a set $C(v_i)$.

Since all algorithms will be learning functions or objectives over input-output pairs, we assume the availability of a joint feature function $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^m$ that computes an m dimensional feature vector for any input-output pair. A feature function can be further decomposed to smaller basic components. For example, in sequence labeling problems, we can define a unary feature function $\Phi_1(x_i, y_i)$ that captures the consistency between each individual x_i and y_i, and a pairwise feature function $\Phi_2(y_i, y_{i+1})$ that captures pairwise compatibility between successive labels. The overall feature function is as following:

$$\Phi(x, y) = \underbrace{\sum_i \Phi_1(x_i, y_i)}_{\text{unary features}} \circ \underbrace{\sum_i \Phi_2(y_i, y_{i+1})}_{\text{pairwise features}} \tag{2.1}$$

The sum above are vector sum, and notation $\circ$ means vector concatenation. A scoring function $f(\Phi; \theta)$ takes this feature vector and the weights θ as input, and scores a candidate structured output y given a structured input x. Given such a scoring function and a new input x, the output computation then involves finding the maximum scoring output:

$$\hat{y} = \operatorname{argmax} f(\Phi(x, y); \theta)$$

In the context of graphical models, each input or output variable is a random variable *node*. If there is a basis feature function over a set of variables, the set of corresponding nodes is called a *clique*, and this basis feature function is called a *factor* connected with this clique. The size of a factor is the number nodes that it contains, and we define the maximum factor size as the feature *complexity*. This is critical to the structured prediction inference. Because scoring function takes the feature function output as input, and changing the value of an output variable will only affects the factors that contain the variable, this factor size determines how far away the changing can be propagated. In some inference algorithms (e.g., variable elimination), the computational complexity depends exponentially on this maximum factor size. An extreme case would be, for example, each factor contains a single output variable and the input. In this case, the problem boils down to a multi-class classification problem since the prediction of any variable v_i would never be able to affect the prediction of another variable v_j. We will show some example tasks of different applications.

Example 1: Part-of-Speech (POS) Tagging Task. Each structured input is a sequence of words. Each output variable v_i stands for POS tag of a word. $C(v_i)$ is the list of all candidate POS tags. Hamming loss (number of incorrect POS tags) is typically employed as loss function. Joint features include unary features (representing words and their POS tags as in a multi-class classifier) and structural features (e.g., label pairs to capture the compatibility of successive labels).

Example 2: Image Labeling Task. Each structured input is an image. Each output variable v_i corresponds to a semantic label of one pixel in the image and $C(v_i)$ is the list of all candidate labels. Intersection-over-Union (IoU) loss – similarity between the predicted region and the ground-truth region for a given semantic labeling – is employed as loss function. Unlike Hamming loss, IoU loss does not decompose into the losses of individual output variables. Joint features include unary features and structural features, e.g., context features that count the co-occurrences of different labels in different spatial relationships such as left, right, above, and below. Intuitively, we are capturing, for example, whether a pixel labeled "sky" is below another pixel labeled "grass."

Example 3: Coreference Resolution Task. Each structured input is a sequence of noun phrases, named *mentions*, extracted from a document in the order of their occurrence in the document. Each output variable v_{ij} $(i < j)$ corresponds to a coreferent linking edge that connects to a pair of mentions. $C(v_{ij})$ only contains two values: true and false indicating coreferent and non-coreferent relations respectively. Pairwise hamming loss which measures the proportion of wrong labeled linking edges can be employed as the output loss function. There are also other task-specific non-decomposable losses, e.g., MUC, BCube, CEAF[1], etc. that directly measures the clustering error. Coreference features include mention pair features (features that capture the similarity of two mentions, e.g., same string, same number, etc.) and cluster features (features about the consistency of labels of mentions within each coreferent cluster.

We structure the existing structured prediction literature along two dimensions: non-search-based vs. search-based frameworks and classical vs. neural network models.

2.2 Non-Search-Based Structured Prediction

The structured prediction problem has interested researchers for a long time, even before the emergence of deep learning. An important difference between structured prediction learning and other model learning is that its training algorithm usually requires to invoke the inference for computing argmin_y multiple times. Thus the efficiency of inference is more critical and challenging in SP learning algorithms.

In the probabilistic graphical modeling framework, a distribution over the variables in the domain is represented as a product of multiple factors based on the structure of a graph. In conditional random fields (CRFs) (Lafferty *et al.*, 2001a), a potential function is learned that represents the conditional distribution $P(y|x)$. The exact inference complexity depends on the maximum factor size in the graph. Structured perceptron and structured SVM (Tsochantaridis *et al.*, 2004) both learn a linear function by optimizing hinge loss except that SSVM learns to maximize a margin for the separating hyperplane while perceptron does not. To overcome the bottleneck of inference speed in training, efforts have been made in improving the efficiency of exact inference (Samdani and Roth, 2012) or approximate inference (Chang *et al.*, 2015c; Ma *et al.*, 2019) during training.

With the development of deep learning, researchers have also combined the classical machine learning models with deep neural networks. The most straightforward way to do this is to replace

[1]Actually these metrics return the accuracy of clustering output. If we want to use them as losses, we need to use the result of 1 minus them.

the linear functions with DNN functions. Deep structured models (DSM) (Chen *et al.*, 2015) can be viewed as an extension of CRFs where the linear potentials are replaced with non-linear functions. In DSMs, the potential functions are decomposed into subfunctions for the nodes and node pairs. The partition function is approximated by loopy belief propagation. Structured prediction energy network (SPEN) (Belanger and McCallum, 2016; Belanger *et al.*, 2017) further extends the pairwise potentials to higher order label interactions through non-linear transformation of label vectors. The energy function returns a value indicating how compatible the output is given the input, which is very similar to the cost function but with a negative sign in the front. Training of the SPENs is similar to that of structured SVMs. Deep value network (DVN) (Gygli *et al.*, 2017) is inspired by policy learning. DVN learns a value function by doing regression over the true value function (negative of loss function). In DVN, each training example will generate one data point for regression training, and the value function is learned by optimizing a cross entropy regression loss over each mini-batch. Our regression based cost function learning is inspired by this work. Both of the two works above use the gradient based inference to compute $\mathrm{argmin}_{y \in \mathcal{Y}}$.

Note that the gradient based inference is very similar to the greedy search if we treat each gradient descent step as an action. Besides the gradient based inference, (Tu and Gimpel, 2018) proposed the generative inference network, in which a network is learned particularly to generate the outputs as approximate inference instead of doing gradient descent, and both inference time and output quality improved.

2.3 Search-Based Frameworks

When talking about search-based approaches for structural prediction, we usually base our analysis on several properties: complete/partial output search state, search space design, search algorithm, update procedure, and scoring function model.

Complete/Partial Output State. This property is related to how we define a search state for the search-based structured prediction. The possible state spaces can be briefly classified into two classes: partial output space, or complete output space. Usually, a search state will include the input x and an output labeling $\hat{y}$. Figure 2.1 shows an example of solving hand-writing recognition problem with partial and complete output space search.

For the partial output space, only part of the output variables are assigned labels, and the remaining variables are left with a default value of “unknown”, while for the complete output space, all the variables need to have an assigned label. Both of these formulations have their advantages. For partial output space, the branching factor is usually small, so that the inference would be fast, but it is only suitable for problems whose input and output has a sequential ordering. Note that when the structured output can be predicted with a sequence of dependent actions, it is easy to apply the imitation learning or reinforcement learning techniques in this case. On the other hand, for complete output space, it is easier to extract higher order features or representations, or apply non-decomposable loss for the input output pairs. But the price for this is that its branching factor might be larger because of the permutation of variable ordering. Also, the search requires an initial output (as the starting point of a search), and usually the search result would be sensitive to this initial output.

Partial output space search has been studied for more than ten years. Inspired by the imitation learning, (Daumé and Marcu, 2005b) proposes beam search optimization framework (LaSO). LaSO formulates the structured prediction as a sequential decision problem where the output can be predicted with a sequence of actions, and learns a heuristic function to guide the beam search to find the path to the ground truth output. The heuristic is usually linear, and is updated along the way when the ground truth output has become unreachable in the current path. For complete output space search, (Doppa *et al.*, 2012) discusses learning of a cost function under different search spaces. The $\mathcal{HC}$-Search further decomposes the original loss into two parts – generation loss and selection loss – and learns two functions for optimizing them (Doppa *et al.*, 2013a).

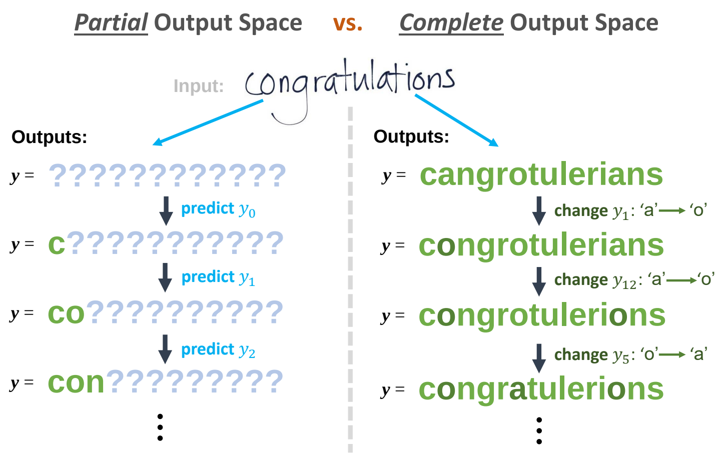


Figure 2.1: Hand-writing recognition example with partial (left) and complete (right) output space search.

Search Space Design. A search space contains five components: search states, actions, successor function, initial state and terminal state. We focus more on actions and successor functions here. The definition of actions and successor function directly determines what operator or change we can apply on a state at each step, and therefore also determines the branching factor and the time complexity of the search algorithm. Beside the simplest case, one can also define "macro actions"(Lam *et al.*, 2015) that allow indirect (e.g., limited discrepancy space action (Doppa *et al.*, 2012; Doppa, 2014)) and more complicated changes to the output.

Search Algorithms. The most common search algorithm for structured prediction is best-first beam search. With fixed size beam, the beam search could limit the expanded search tree (number of states) in $O(bd)$ where d is the depth of the search tree and b is the branching factor, but also allow the search to have a chance of correcting the current paths when the error occurs. When beam size equals 1, the beam search reduces to the greedy search.

We want to emphasize one more aspect of greedy search. For any problem in structured prediction, if we learn a policy or a recurrent classifier to predict the output incrementally with greedy search, all techniques in imitation learning will be applicable. This is usually applied together with the partial output states. SEARN(Daumé *et al.*, 2009), DAgger(Ross *et al.*, 2011), and LOLS(Chang *et al.*, 2015a) are the frameworks in this formulation.

Weight Update. This property is about when to perform weight update during search. We have 3 candidate choices: update during search, update after each example, update after each mini-batch. In other words, we care about whether the search itself is transparent to the optimizer or not. This is because, due to the formulation of search space, search-based algorithm is doing structure prediction in the form of "learning for inference" rather than "learning with inference". Therefore, compared to the blackbox inference, the search procedure itself contains more information useful for learning. On the other hand, we can always aggregate the training data during search, and perform the update after the search on each example. This can be further delayed till the end of example mini-batch.

When to update is critical for some methods. For example, in beam search optimization, (Collins and Roark, 2004), (Huang *et al.*, 2012) and (Björkelund and Kuhn, 2014) all apply "update after search", but use *early update*, *max-violation update* and *delayed update* respectively in order to achieve different search behaviors. On the other hand, LaSO employs update during search, but with error condition. The definition of search error could control the learning to perform a relatively conservative or aggressive update in search.

Scoring Function Model. Scoring function model can evaluate a search state, or a state-action pair, so that the search procedure can be guided. This function can be used as the heuristic function or cost function in A* search or beam search. The most traditional search-based approaches use linear scoring functions. Recent works have explored replacing the linear models with deep neural network models. For example, for LaSO mentioned above, (Wiseman and M. Rush, 2016) further extends this work by combining beam search optimization with the Seq2seq model (Sutskever *et al.*, 2014), and replacing the linear function with RNN model.

2.4 Summary of Existing Works

As shown in Table 2.1, we can classify most of the structured prediction approaches into this four cell table. The left column is about traditional linear methods, and the right column is the recent progress with DNN models. The bottom row is about search-based methods, while the top row consists of non-search-based methods. We can place most of the existing works into this four cell table. All the contributions of this thesis will fall into the two cells in the bottom row, i.e., the search-based approaches. More specifically, our work Prune-and-Score and MTSP for entity analysis work is in the bottom-left cell. It is easy to see that the bottom-right cell, which corresponds to the search-based DNN methods, has not been explored thoroughly. Our new framework contributes to the bottom-right cell.

	Non-Deep Models	**Deep Models**
Non-Search-based	CRF, SSVM, SPerceptron	DSM, SPEN, DVN, InfNet
Search-based	LaSO, Prune&Score, MTSP HC-Search	Seq2Seq-BSO

Table 2.1: Structural prediction approaches.

Taking a closer look at the bottom row of Table 2.1, we can further classify the search-based approaches with the properties we listed in the last section as shown in Table 2.2. We split the table into two parts. The upper part corresponds to classical models, while lower part to the DNN models. Our Prune-and-Score work uses partial space while MTSP is in complete space, but neither of them uses DNN models. Therefore, they are in the upper part of the table. MTSP is special because it is dealing with multiple tasks. Our last contribution is a new complete output space search algorithm guided by DNN based scoring functions, which corresponds to the last row in the table.

	State Space	**Scoring Models**	**Update**	**Search Algm.**
LaSO	Partial	Linear	During Search	Beam Search
Prune&Score	Partial	DecisionTree	After Search	Greedy Search
MTSP	Complete	Linear	After Search	Beam Search
HC-Search	Complete	Linear	After Search	Beam Search
Seq2Seq-BSO	Partial	DNN	During Search	Beam Search
HC-Nets	Complete	DNN	After Mini-batch	Beam Search

Table 2.2: Search-based structured prediction approaches.

Integrating deep learning advances into structured prediction frameworks raise several challenges. We need to ensure the stability and robustness of training and inference process. Existing works (Belanger and McCallum, 2016; Belanger *et al.*, 2017; Gygli *et al.*, 2017) employ gradient-based inference in the relaxed continuous space followed by rounding to compute the final solution. This procedure involves a large set of interdependent parameters, including the initial output, step size, number of steps, and a rounding threshold to convert real-valued output variables back to discrete space. These parameters contribute to the lack of robustness in training/inference when employing gradient-based inference. Moreover, the relaxation of structured outputs to continuous space make it difficult to enforce domain-specific constraints. The only existing work (Lee *et al.*, 2019) relies on Lagrangian multiplier in the objective to perform constrained optimization. In our proposed $\mathcal{HC}$-Nets framework, we try to provide a general solution to overcome all the above-mentioned challenges.

Chapter 3: Prune-and-Score: Learning Greedy Policies for Structured Prediction with Application to Coreference Resolution

Greedy policy based structured prediction (SP) methods are very successful in practice (Stoyanov and Eisner, 2012; Chang *et al.*, 2013; Durrett and Klein, 2013; Xu *et al.*, 2009; Sutskever *et al.*, 2014; Wiseman and M. Rush, 2016; Kim *et al.*, 2017). The key idea is to pose SP as a sequential decision-making problem over a fixed ordering of the inter-dependent output variables. However, this family of methods can be potentially sub-optimal in terms of accuracy when the size of the structured outputs (number of output variables) and the number of candidate labels for each output variable is very large. In this Chapter, we develop a search-based learning approach called "*Prune-and-Score*" to improve the accuracy of greedy policy based SP methods for such hard tasks. The key idea is to learn a pruning function that prunes bad decisions and a scoring function that then selects the best among the remaining decisions. We show the efficacy of this approach for coreference resolution, i.e., clustering mentions that refer to the same entity, which is a hard NLP task.

Coreference resolution is the task of clustering a set of mentions in the text such that all mentions in the same cluster refer to the same entity. It is one of the first stages in deep language understanding and has a big potential impact on the rest of the stages. Several of the state-of-the-art approaches learn a scoring function defined over mention pairs, cluster-mention or cluster-cluster pairs to guide the coreference decision-making process (Daumé III, 2006; Bengtson and Roth, 2008; Rahman and Ng, 2011b; Stoyanov and Eisner, 2012; Chang *et al.*, 2013; Durrett *et al.*, 2013; Durrett and Klein, 2013). One common and persistent problem with these approaches is that the scoring function has to make all the coreference decisions, which leads to a highly non-realizable learning problem.

Inspired by the $\mathcal{HC}$-Search Framework (Doppa *et al.*, 2014a) for studying a variety of structured prediction problems (Lam *et al.*, 2013; Doppa *et al.*, 2014c), we study a novel approach for search-based coreference resolution called *Prune-and-Score*. $\mathcal{HC}$-Search is a divide-and-conquer solution that learns multiple components with pre-defined roles, and each of them contribute towards the overall goal by making the role of the other components easier. The $\mathcal{HC}$-Search framework operates in the space of complete outputs, and relies on the loss function which is only defined on the complete outputs to drive its learning. Unfortunately, this method does not work for incremental coreference resolution since the search space for coreference resolution consists of partial outputs, i.e., a set of mentions only some of which have been clustered so far.

We develop an alternative framework to $\mathcal{HC}$-Search that allows us to effectively learn from partial output spaces and apply it to greedy coreference resolution. The key idea of our work is to address the problem of non-realizability of the scoring function by learning two different functions: 1) a *pruning function* to prune most of the bad decisions, and 2) a *scoring function* to pick the best decision among those that are remaining. Our Prune-and-Score approach is a particular instantiation of the general idea of learning nearly-sound constraints for pruning, and leveraging the learned constraints to learn improved heuristic functions for guiding the search. The pruning constraints can take different forms (e.g., classifiers, decision-lists, or ranking functions) depending on the search architecture. Therefore, other coreference resolution systems (Chang *et al.*, 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014) can also benefit from this idea. While our basic idea of two-level selection might appear similar to the cascades or coarse-to-fine inference architectures (Felzenszwalb and McAllester, 2007; Weiss and Taskar, 2010), the details differ significantly. Importantly, our pruning and scoring functions operate sequentially at each greedy search step, whereas in the cascades approach, the second level function makes its prediction only after the first level decision-making is done.

Summary of Contributions. The main contributions of our work are as follows. First, we motivate and introduce the *Prune-and-Score* approach to search-based coreference resolution. Second, we identify a decomposition of the overall loss of the Prune-and-Score approach into the pruning loss and the scoring loss, and reduce the problem of learning these two functions to rank learning, which allows us to leverage powerful and efficient off-the-shelf rank learners. Third, we evaluate our approach on OntoNotes, ACE, and MUC data, and show that it compares favorably to several state-of-the-art approaches as well as a greedy search-based approach that uses a single scoring function.

3.1 Related Work

The work on learning-based coreference resolution can be broadly classified into three types. First, the *pair-wise classifier* approaches learn a classifier on mention pairs (edges) (Soon *et al.*, 2001; Ng and Cardie, 2002; Bengtson and Roth, 2008), and perform some form of approximate decoding or post-processing using the pair-wise scores to make predictions. However, the pairwise classifier approach suffers from several drawbacks including class imbalance (fewer positive edges compared to negative edges) and not being able to leverage the global structure (instead making independent local decisions).

Second, the *global* approaches such as Structured SVMs and Conditional Random Fields (CRFs) learn a cost function to score a potential clustering output for a given input set of mentions (Mccallum and Wellner, 2003; Finley and Joachims, 2005; Culotta *et al.*, 2007; Yu and Joachims, 2009; Haghighi and Klein, 2010; Wick *et al.*, 2011, 2012; Fernandes *et al.*, 2012). These methods address some of the problems with pair-wise classifiers, however, they suffer from the intractability of “Argmin” inference (finding the least cost clustering output among exponential possibilities) that is encountered during both training and testing. As a result, they resort to approximate inference algorithms (e.g., MCMC, loopy belief propagation), which can suffer from local optima.

Third, the *incremental* approaches construct the clustering output incrementally by processing the mentions in some order (Daumé III, 2006; Denis and Baldridge, 2008; Rahman and Ng, 2011b; Stoyanov and Eisner, 2012; Chang *et al.*, 2013; Durrett *et al.*, 2013; Durrett and Klein, 2013). These methods learn a scoring function to guide the decision-making process and differ in the form of the scoring function (e.g., mention pair, cluster-mention or cluster-cluster pair) and how it is being learned. They have shown great success and are very efficient. Indeed, several of the approaches that have achieved state-of-the-art results on OntoNotes fall under this category (Chang *et al.*, 2013; Durrett *et al.*, 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014). However, their efficiency requirement leads to a highly non-realizable learning problem. Our Prune-and-Score approach is complementary to these methods, as we show that having a pruning function (or a set of learned pruning rules) makes the learning problem easier and can improve over the performance of scoring-only approaches. Also, the models in (Chang *et al.*, 2013; Durrett *et al.*, 2013) try to leverage cluster-level information implicitly (via latent antecedents) from mention-pair features, whereas our model explicitly leverages the cluster level information.

Coreference resolution systems can benefit by incorporating the world knowledge including rules, constraints, and additional information from external knowledge bases (Lee *et al.*, 2013; Rahman and Ng, 2011a; Ratinov and Roth, 2012; Chang *et al.*, 2013; Zheng *et al.*, 2013; Hajishirzi *et al.*, 2013). Our work is orthogonal to this line of work, but domain constraints and rules can be incorporated into our model as done in (Chang *et al.*, 2013).

3.2 Problem Setup

Coreference resolution is a structured prediction problem where the set of mentions $m_1, m_2, \cdots, m_D$ extracted from a document correponds to a structured input x and the structured output y corresponds to a partition of the mentions into a set of clusters $C_1, C_2, \cdots, C_k$. Each mention m_i belongs to exactly one of the clusters C_j. We are provided with a training set of input-output pairs drawn from an unknown distribution $\mathcal{D}$, and the goal is to return a function/predictor from inputs to outputs. The learned predictor is evaluated against a non-negative *loss function* $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re^+$, $L(x, y', y)$ is the loss associated with predicting incorrect output y' for input x when the true output is y (e.g., B-Cubed Score).

In this work, we formulate the coreference resolution problem in a search-based framework. There are three key elements in this framework: 1) the *Search space* $\mathcal{S}_p$ whose states correspond to partial clustering outputs; 2) the *Action pruning function* $\mathcal{F}_{prune}$ that is used to prune irrelevant actions at each state; and 3) the *Action scoring function* $\mathcal{F}_{score}$ that is used to construct a complete clustering output by selecting actions from those that are left after pruning. $\mathcal{S}_p$ is a 3-tuple $\langle I, A, T \rangle$, where I is the initial state function, A gives the set of possible actions in a given state, and T is a predicate which is true for terminal states. In our case, $s_0 = I(x)$ corresponds to a state where every mention is unresolved, and $A(s_i)$ consists of actions to place the next mention m_{i+1} in each cluster in s_i or a NEW action which creates a new cluster for it. Terminal nodes correspond to states with all mentions resolved.

We focus on greedy search. The decision process for constructing an output corresponds to selecting a sequence of actions leading from the initial state to a terminal state using both $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$, which are parameterized functions over state-action pairs ($F_{prune}(\phi_1(s, a)) \in \Re$ and $F_{score}(\phi_2(s, a)) \in \Re$), where ϕ_1 and ϕ_2 stand for feature functions. We want to learn the parameters of both $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ such that the predicted outputs on unseen inputs have low expected loss.

3.3 Greedy Prune-and-Score Approach

Our greedy *Prune-and-Score* approach for coreference resolution is parameterized by a pruning function $\mathcal{F}_{prune} : \mathcal{S} \times \mathcal{A} \mapsto \Re$, a scoring function $\mathcal{F}_{score} : \mathcal{S} \times \mathcal{A} \mapsto \Re$, and a pruning parameter $b \in [1, A_{max}]$, where A_{max} is the maximum number of actions at any state $s \in \mathcal{S}$. Given a set of input mentions $m_1, m_2, \cdots, m_D$ extracted from a document (input x), and a pruning parameter b, our Prune-and-Score approach makes predictions as follows. The search starts at the initial

Algorithm 1 Greedy Prune-and-Score Resolver

Input: x = set of mentions $m_1, m_2, \cdots, m_D$ from a document D,
$\langle I, A, T \rangle$ = Search space definition, $\mathcal{F}_{prune}$ = learned pruning function,
b = pruning parameter, $\mathcal{F}_{score}$ = learned scoring function
Output: y, the coreference output

```
1: $s \leftarrow I(x)$ // initial state
2: while not $T(s)$ do
3:     $A^{'} \leftarrow$ Top $b$ actions from $A(s)$ according to $\mathcal{F}_{prune}$ // prune
4:     $a_p \leftarrow \arg\max_{a \in A'} \mathcal{F}_{score}(s, a)$ // score
5:     $s \leftarrow$ Apply $a_p$ on $s$
6: end while
7: return coreference output corresponding to $s$
```

state $s_0 = I(x)$ (see Algorithm 1). At each non-terminal state s, the pruning function $\mathcal{F}_{prune}$ retains only the top b actions (A') from $A(s)$ (Step 3), and the scoring function $\mathcal{F}_{score}$ picks the best scoring action $a_p \in A'$ (Step 4) to reach the next state. When a terminal state is reached its contents are returned as the prediction. Figure 3.1 illustrates the decision-making process of our Prune-and-Score approach for an example state.

We now formalize the learning objective of our Prune-and-Score approach. Let $\hat{y}$ be the predicted coreference output for a coreference input-output pair (x, y^*). The expected loss of the greedy Prune-and-Score approach $\mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$ for a given pruning function $\mathcal{F}_{prune}$ and scoring function $\mathcal{F}_{score}$ can be defined as follows.

$$\mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score}) = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ L\left(x, \hat{y}, y^*\right)$$

Our goal is to learn an optimal pair of pruning and scoring functions $\left(\mathcal{F}^o_{prune}, \mathcal{F}^o_{score}\right)$ that minimizes the expected loss of the Prune-and-Score approach. The behavior of our Prune-and-Score approach depends on the pruning parameter b, which dictates the workload of pruning and scoring functions. For small values of b (aggressive pruning), pruning function learning may be harder, but scoring function learning will be easier. Similarly, for large values of b (conservative pruning), scoring function learning becomes hard, but pruning function learning is easy. Therefore, we would expect beneficial behavior if pruning function can aggressively prune (small values of b) with little loss in accuracy. It is interesting to note that our Prune-and-Score approach degenerates to existing incremental approaches that use only the scoring function for search (Daumé III, 2006; Rahman and Ng, 2011b) when $b = \infty$. Additionally, for $b = 1$, our pruning function coincides with the scoring function.

Analysis of Representational Power. The following proposition formalizes the intuition that two functions are strictly better than one in expressive power. See Appendix for the proof.

Proposition 1 *Let $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ be functions from the same function space. Then for all learning problems, $\min_{\mathcal{F}_{score}} \mathcal{E}(\mathcal{F}_{score}, \mathcal{F}_{score}) \geq \min_{(\mathcal{F}_{prune}, \mathcal{F}_{score})} \mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$. Moreover there exist learning problems for which $\min_{\mathcal{F}_{score}} \mathcal{E}(\mathcal{F}_{score}, \mathcal{F}_{score})$ can be arbitrarily worse than $\min_{(\mathcal{F}_{prune}, \mathcal{F}_{score})} \mathcal{E}(\mathcal{F}_{prune}, \mathcal{F}_{score})$.*

Proof. The first part of the proposition follows from the fact that the first minimization is over a subset of the choices considered by the second. For the second part, consider a problem with a single training instance with search space shown in Figure 3.3. We assume linear $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ functions of features $\Phi(n)$, where n is an action. The highlighted nodes correspond to the target path. The Prune-and-Score approach with $b = 2$ can find $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ functions that are consistent with the target path. For example, with $\mathcal{F}_{prune} = (1, 0)$ and $\mathcal{F}_{score} = (1, 2)$ and pruning parameter 2 Prune-and-Score can achieve zero loss on this problem. However, it can be verified that there is no set of weights that satisfies all the constraints for imitating the target path by the Scoring-Only approach ($\mathcal{F}_{score}(3) > \mathcal{F}_{score}(2)$ and $\mathcal{F}_{score}(8) > \mathcal{F}_{score}(7)$ in particular).

3.4 Learning Algorithms

In general, learning the optimal $\left(\mathcal{F}^{o}_{prune}, \mathcal{F}^{o}_{score}\right)$ pair can be intractable due to their potential interdependence. Specifically, when learning $\mathcal{F}_{prune}$ in the worst case there can be ambiguity about which of the non-optimal actions to retain, and for only some of those an effective $\mathcal{F}_{score}$ can be found. However, we observe a loss decomposition in terms of the individual losses due to $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$, and develop a stage-wise learning approach that first learns $\mathcal{F}_{prune}$ and then learns a corresponding $\mathcal{F}_{score}$.

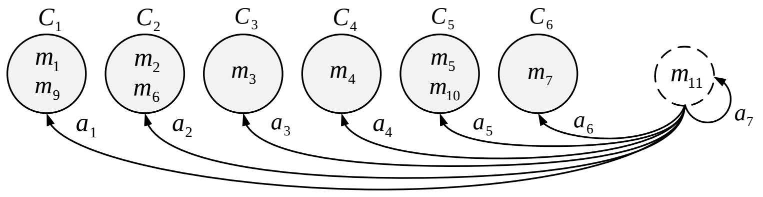
3.4.1 Loss Decomposition

The overall loss of the *Prune-and-Score* approach $\mathcal{E}\left(\mathcal{F}_{prune}, \mathcal{F}_{score}\right)$ can be decomposed into *pruning loss* ϵ_{prune}, the loss due to $\mathcal{F}_{prune}$ not being able to retain the *optimal* terminal state in the search space; and *scoring loss* $\epsilon_{score|\mathcal{F}_{prune}}$, the additional loss due to $\mathcal{F}_{score}$ not guiding the

(a) Text with input set of mentions

Ramallah (West Bank $_2$)$_1$ 10-15 (AFP$_3$) - Eyewitnesses$_4$ reported that Palestinians$_5$ demonstrated today Sunday in the West Bank$_6$ against the Sharm el-Sheikh$_7$ summit to be held in Egypt$_8$ tomorrow Monday. In Ramallah$_9$, around 500 people$_{10}$ took to **the town**$_{11}$'s streets chanting slogans denouncing the summit ...

(b) Illustration of Prune-and-Score approach



State: $s = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ Actions: $A(s) = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$

Pruning step:

$$\underbrace{\boxed{\underset{2.5}{a_2}\quad \underset{2.2}{a_1}\quad \underset{1.9}{a_7}}}_{b=3}\quad \underset{1.5}{a_5}\quad \underset{1.4}{a_6}\quad \underset{0.7}{a_3}\quad \underset{0.4}{a_4} \longleftarrow \mathcal{F}_{prune} \text{ values}$$

$A'(s) = \{a_2, a_1, a_7\}$

Scoring step:

$$\underset{4.5}{a_1}\quad \underset{3.1}{a_2}\quad \underset{2.6}{a_7} \longleftarrow \mathcal{F}_{score} \text{ values}$$

Decision: a_1 is the best action for state s

Figure 3.1: Illustration of Prune-and-Score approach. (a) Text with input set of mentions. Mentions are highlighted and numbered. (b) Illustration of decision-making process for mention m_{11}. The partial clustering output corresponding to the current state s consists of six clusters denoted by $C_1, C_2, \cdots, C_6$. Highlighted circles correspond to the clusters. Edges from mention m_{11} to each of the six clusters and to itself stand for the set of possible actions $A(s)$ in state s, and are denoted by $a_1, a_2, \cdots, a_7$. The pruning function $\mathcal{F}_{prune}$ scores all the actions in $A(s)$ and only keeps the top 3 actions $A' = \{a_2, a_1, a_7\}$ as specified by the pruning parameter b. The scoring function picks the best scoring action $a_1 \in A'$ as the final decision, and mention m_{11} is merged with cluster C_1.

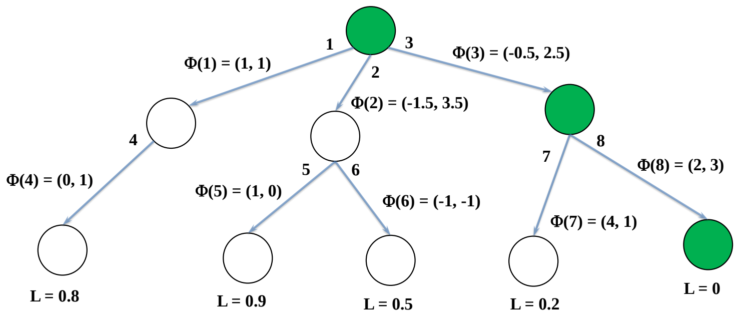


Figure 3.2: An example that illustrates that methods that use only scoring function for search can suffer arbitrary large loss compared to Prune-and-Score approach.

greedy search to the best terminal state after pruning using $\mathcal{F}_{prune}$. Below, we will define these losses more formally.

Pruning Loss is defined as the expected loss of the *Prune-and-Score* approach when we perform greedy search with $\mathcal{F}_{prune}$ and $\mathcal{F}^*_{score}$, the optimal scoring function. A scoring function is said to be *optimal* if at every state s in the search space $\mathcal{S}_p$, and for any set of remaining actions $A(s)$, it can score each action $a \in A(s)$ such that greedy search can reach the best terminal state (as evaluated by task loss function L) that is reachable from s through $A(s)$. Unfortunately, computing the optimal scoring function is highly intractable for the non-decomposable loss functions that are employed in coreference resolution (e.g., B-Cubed F1). The main difficulty is that the decision at any one state has interdependencies with future decisions (see Section 5.5 in (Daumé III, 2006) for more details). So we need to resort to some form of *approximate* optimal scoring function that exhibits the intended behavior. This is very similar to the dynamic oracle concept developed for dependency parsing (Goldberg and Nivre, 2013).

Let y^*_{prune} be the coreference output corresponding to the terminal state reached from input x by *Prune-and-Score* approach when performing search using $\mathcal{F}_{prune}$ and $\mathcal{F}^*_{score}$. Then the pruning loss can be expressed as follows.

$$\epsilon_{prune} = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, y^*_{prune}, y^*\right)$$

Scoring Loss is defined as the additional loss due to $\mathcal{F}_{score}$ not guiding the greedy search to the best terminal state reachable via the pruning function $\mathcal{F}_{score}$ (i.e., y^*_{prune}). Let $\hat{y}$ be the coreference output corresponding to the terminal state reached by *Prune-and-Score* approach by performing search with $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}$ for an input x. Then the scoring loss can be expressed as follows:

$$\epsilon_{score|\mathcal{F}_{prune}} = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ L\left(x, \hat{y}, y^*\right) - L\left(x, y^*_{prune}, y^*\right)$$

The overall loss decomposition of our *Prune-and-Score* approach can be expressed as follows.

$$\begin{aligned}\mathcal{E}\left(\mathcal{F}_{prune}, \mathcal{F}_{score}\right) = &\underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ L\left(x, y^*_{prune}, y^*\right)}_{\epsilon_{\mathbf{prune}}} + \\ &\underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ L\left(x, \hat{y}, y^*\right) - L\left(x, y^*_{prune}, y^*\right)}_{\epsilon_{\mathbf{score}|\mathcal{F}_{\mathbf{prune}}}}\end{aligned}$$

3.4.2 Stage-wise Learning

The loss decomposition motivates a learning approach that targets minimizing the errors of pruning and scoring functions independently. In particular, we optimize the overall loss of the *Prune-and-Score* approach in a stage-wise manner. We first train a pruning function $\hat{\mathcal{F}}_{prune}$ to optimize the pruning loss component ϵ_{prune} and then train a scoring function $\hat{\mathcal{F}}_{score}$ to optimize the scoring loss $\epsilon_{score|\hat{\mathcal{F}}_{prune}}$ conditioned on $\hat{\mathcal{F}}_{prune}$.

$$\begin{aligned}\hat{\mathcal{F}}_{prune} &\approx \arg\min_{\mathcal{F}_{prune}\in\mathbf{F_p}}\ \ \epsilon_{prune}\\ \hat{\mathcal{F}}_{score} &\approx \arg\min_{\mathcal{F}_{score}\in\mathbf{F_s}}\ \ \epsilon_{score|\hat{\mathcal{F}}_{prune}}\end{aligned}$$

Note that this approach is myopic in the sense that $\hat{\mathcal{F}}_{prune}$ is learned without considering the implications for learning $\hat{\mathcal{F}}_{score}$. Below, we first describe our approach for pruning function learning, and then explain our scoring function learning algorithm.

3.4.3 Pruning Function Learning

In our greedy *Prune-and-Score* approach, the role of the pruning function $\mathcal{F}_{prune}$ is to prune away irrelevant actions (as specified by the pruning parameter b) at each search step. More specifically, we want $\mathcal{F}_{prune}$ to score actions $A(s)$ at each state s such that the optimal action $a^* \in A(s)$ is ranked within the top b actions to minimize ϵ_{prune}. For this, we assume that for any training input-output pair (x, y^*) there exists a unique action sequence, or *solution path* (initial state to terminal state), for producing y^* from x. More formally, let $(s_0^*, a_0^*), (s_1^*, a_1^*), \cdots, (s_D^*, \varnothing)$ correspond to the sequence of state-action pairs along this solution path, where s_0^* is the initial state and s_D^* is the terminal state. The goal is to learn the parameters of $\mathcal{F}_{prune}$ such that at each state s_i^*, $a_i^* \in A(s_i^*)$ is ranked among the top b actions.

While we can employ an *online-LaSO* style approach (Daumé and Marcu, 2005b; Xu *et al.*, 2009) to learn the parameters of the pruning function, it is quite inefficient, as it must regenerate the same search trajectory again and again until it learns to make the right decision. Additionally, this approach limits applicability of the off-the-shelf learners to learn the parameters of $\mathcal{F}_{prune}$. To overcome these drawbacks, we apply offline training.

Reduction to Rank Learning. We reduce the pruning function learning to a rank learning problem. This allows us to leverage powerful and efficient off-the-shelf rank-learners (Liu, 2009). The reduction is as follows. At each state s_i^* on the solution path of a training example (x, y^*), we create an example by labeling optimal action $a_i^* \in A(s_i^*)$ as the only relevant action, and then try to learn a ranking function that can rank actions such that the relevant action a_i^* is in the top b actions, where b is the input pruning paramter. In other words, we have a rank learning problem, where the learner's goal is to optimize the *Precision at Top-b*. The training approach creates such an example for each state s in the solution path. The set of aggregate imitation examples collected over all the training data is then given to a rank learner (e.g., LambdaMART (Burges, 2010)) to learn the parameters of $\mathcal{F}_{prune}$ by optimizing the *Precision at Top-b* loss. See Algorithm 2 for the pseudocode.

If we can learn a function $\mathcal{F}_{prune}$ that is consistent with these imitation examples, then the learned pruning function is guaranteed to keep the solution path within the pruned space for all the training examples. We can also employ more advanced imitation learning algorithms including DAgger (Ross *et al.*, 2011) and SEARN (Hal Daumé III *et al.*, 2009) if we are provided with an (approximate) optimal scoring function $\mathcal{F}_{score}^*$ that can pick optimal actions at states that are not in the solution path (i.e., *off-trajectory* states).

3.4.4 Scoring Function Learning

Given a learned pruning function $\mathcal{F}_{prune}$, we want to learn a scoring function that can pick the best action from the b actions that remain after pruning at each state. We formulate this problem in the framework of *imitation learning* (Khardon, 1999). More formally, let

Algorithm 2 Pruning Function Learning

```
Input: $\mathcal{D}$ = Training data, $(I, A, T)$ = Search space, $b$ = Pruning parameter
 1: Initialize the set of ranking examples $\mathcal{R} = \emptyset$
 2: for each training example $(x, y^*) \in \mathcal{D}$ do
 3:     $s \leftarrow I(x)$ // initial state
 4:     while not $T(s)$ do
 5:         Generate an example $R_t$ to imitate this search step
 6:         Aggregate training data: $\mathcal{R} = \mathcal{R} \cup R_t$
 7:         $s \leftarrow$ Apply $a^*$ on $s$
 8:     end while
 9: end for
10: $\mathcal{F}_{prune}$ = Rank-Learner($\mathcal{R}$)
11: return pruning function $\mathcal{F}_{prune}$
```

$(\hat{s}_0, a_0^*), (\hat{s}_1, a_1^*), \cdots, (\hat{s}_D^*, \varnothing)$ correspond to the sequence of state-action pairs along the greedy trajectory obtained by running the Prune-and-Score approach with $\mathcal{F}_{prune}$ and $\mathcal{F}_{score}^*$, the optimal scoring function, on a training example (x, y^*), where $\hat{s}_D^*$ is the best terminal state in the pruned space. The goal of our imitation training approach is to learn the parameters of $\mathcal{F}_{score}$ such that at each state $\hat{s}_i$, $a_i^* \in A'$ is ranked higher than all other actions in A', where $A' \subseteq A(\hat{s}_i)$ is the set of b actions that remain after pruning.

It is important to note that the distribution of states in the pruned space due to $\mathcal{F}_{prune}$ on the testing data may be somewhat different from those on training data. Therefore, we train our scoring function via cross-validation by training the scoring function on heldout data that was not used to train the pruning function. This methodology is commonly employed in Re-Ranking and Stacking approaches (Collins, 2000; Cohen and de Carvalho, 2005).

Our scoring function learning procedure uses cross validation and consists of the following four steps. First, we divide the training data $\mathcal{D}$ into k folds. Second, we learn k different pruners, where each pruning function $\mathcal{F}_{prune}^i$ is learned using the data from all the folds excluding the i^{th} fold. Third, we generate ranking examples for scoring function learning as described above using each pruning function $\mathcal{F}_{prune}^i$ on the data it was not trained on. Finally, we give the aggregate set of ranking examples $\mathcal{R}$ to a rank learner (e.g., SVM-Rank or LambdaMART) to learn the scoring function $\mathcal{F}_{score}$. See Algorithm 3 for the pseudocode.

Approximate Optimal Scoring Function. If the learned pruning function is not consistent with the training data, we will encounter states $\hat{s}_i$ that are not on the target path, and we will need

Algorithm 3 Scoring Function Learning via Cross Validation

```
Input: $\mathcal{D}$ = Training data, $\mathcal{S}_p$ = Search space, $b$ = Pruning parameter, $\mathcal{F}^*_{score}$ = Optimal scoring function
1: Divide the training set $\mathcal{D}$ into $k$ folds $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k$
2: // Learn $k$ different pruning functions
3: for $i = 1$ to $k$ do
4:     $T_i = \cup_{j \neq i} \mathcal{D}_j$
5:     $\mathcal{F}^i_{prune}$ = Learn-Pruner$(T_i, \mathcal{S}_p, b)$
6: end for
7: // Generate ranking examples for scoring function training
8: Intialize the set of ranking examples $\mathcal{R} = \emptyset$
9: for $i = 1$ to $k$ do
10:     for each training example $(x, y^*) \in \mathcal{D}_i$ do
11:         $s \leftarrow I(x)$ // initial state
12:         while not Terminal$(s)$ do
13:             $A' \leftarrow$ Top $b$ actions from $A(s)$ according to $\mathcal{F}^i_{prune}$
14:             $a^* \leftarrow \text{argmax}_{a \in A'} \mathcal{F}^*_{score}(s, a)$
15:             Generate ranking example $R_t$ to imitate this search step
16:             Aggregate training data: $\mathcal{R} = \mathcal{R} \cup R_t$
17:             $s \leftarrow$ Apply $a^*$ on $s$
18:         end while
19:     end for
20: end for
21: $\mathcal{F}_{score}$ = Rank-Learner($\mathcal{R}$)
22: return scoring function $\mathcal{F}_{score}$
```

some supervision for learning in those cases. As discussed before in Section 3.4.1, computing an optimal scoring function $\mathcal{F}^*_{score}$ is intractable for combinatorial loss functions that are used for coreference resolution. So we employ an approximate function from existing work that is amenable to evaluate partial outputs (Daumé III, 2006). It is a variant of the ACE scoring function that removes the bipartite matching step from the ACE metric. Moreover this score is computed only on the partial coreference output corresponding to the "after state" s' resulting from taking action a in state s, i.e., $\mathcal{F}^*_{score}(s, a) = \mathcal{F}^*_{score}(s')$. To further simplify the computation, we give uniform weight to the three types of costs: 1) Credit for correct linking, 2) Penalty for incorrect linking, and 3) Penalty for missing links. Intuitively, this is similar to the correct-link count computed only on a subgraph. We direct the reader to (Daumé III, 2006) for more details (see Section 5.5).

3.5 Experiments and Results

In this section, we evaluate our greedy Prune-and-Score approach on three benchmark corpora – OntoNotes 5.0 (Pradhan *et al.*, 2012), ACE 2004 (NIST, 2004), and MUC6 (MUC6, 1995) – and compare it against the state-of-the-art approaches for coreference resolution. For OntoNotes data, we report the results on both gold mentions and predicted mentions. We also report the results on gold mentions for ACE 2004 and MUC6 data.

3.5.1 Experimental Setup

Datasets. For OntoNotes corpus, we employ the official split for training, validation, and testing. There are 2802 documents in the training set; 343 documents in the validation set; and 345 documents in the testing set. The ACE 2004 corpus contains 443 documents. We follow the (Culotta *et al.*, 2007; Bengtson and Roth, 2008) split in our experiments by employing 268 documents for training, 68 documents for validation, and 107 documents (ACE2004-CULOTTA-TEST) for testing. We also evaluate our system on the 128 newswire documents in ACE 2004 corpus for a fair comparison with the state-of-the-art. The MUC6 corpus containts 255 documents. We employ the official test set of 30 documents (MUC6-TEST) for testing purposes. From the remaining 225 documents, which includes 195 official training documents and 30 dry-run test documents, we randomly pick 30 documents for validation, and use the remaining ones for training.

Evaluation Metrics. We compute three most popular performance metrics for coreference resolution: MUC (Vilain *et al.*, 1995), B-Cubed (Bagga and Baldwin, 1998), and Entity-based CEAF (CEAF$_{\phi 4}$) (Luo, 2005). As it is commonly done in CoNLL shared tasks (Pradhan *et al.*, 2012), we employ the average F1 score (CoNLL F1) of these three metrics for comparison purposes. We evaluate all the results using the updated version[1] (7.0) of the coreference scorer.

Features. We built[2] our coreference resolver based on the Easy-first coreference system (Stoyanov and Eisner, 2012), which is derived from the Reconcile system (Stoyanov *et al.*, 2010). We essentially employ the same features as in the Easy-first system. However, we provide some high-level details that are necessary for subsequent discussion. Recall that our features $\phi(s, a)$ for both pruning and scoring functions are defined over state-action pairs, where each state s

[1]http://code.google.com/p/reference-coreference-scorers/

[2]See http://research.engr.oregonstate.edu/dral/ for our software.

consists of a set of clusters and an action a corresponds to merging an unprocessed mention m with a cluster C in state s or create one for itself. Therefore, $\phi(s, a)$ defines features over cluster-mention pairs (C, m). Our feature vector consists of three parts: a) mention pair features; b) entity pair features; and c) a single indicator feature to represent NEW action (i.e., mention m starts its own cluster). For mention pair features, we average the pair-wise features over all links between m and every mention m_c in cluster C (often referred to as *average-link*). Note that, we cannot employ the *best-link* feature representation because we perform offline training and do not have weights for scoring the links. For entity pair features, we treat mention m as a singleton entity and compute features by pairing it with the entity represented by cluster C (exactly as in the Easy-first system). The indicator feature will be 1 for the NEW action and 0 for all other actions. We have a total of 140 features: 90 mention pair features; 49 entity pair features; and one NEW indicator feature. We believe that our approach can benefit from employing features of the mention for the NEW action (Rahman and Ng, 2011b; Durrett and Klein, 2013). However, we were constrained by the Reconcile system and could not leverage these features for the NEW action.

Base Rank-Learner. Our pruning and scoring function learning algorithms need a base rank-learner. We employ LambdaMART (Burges, 2010), a state-of-the art rank learner from the RankLib[3] library. LambdaMART is a variant of boosted regression trees. We use a learning rate of 0.1, specify the maximum number of boosting iterations (or trees) as 1000 noting that its actual value is automatically decided based on the validation set, and tune the number of leaves per tree based on the validation data. Once we fix the hyper-parameters of LambdaMART, we train the final model on all of the training data. LambdaMART uses an internal train/validation split of the input ranking examples to decide when to stop the boosting iterations. We fixed this ratio to 0.8 noting that the performance is not sensitive to this parameter. For scoring function learning, we used 5 folds for the cross-validation training.

Pruning Parameter b. The hyper-parameter b controls the amount of pruning in our Prune-and-Score approach. We perform experiments with different values of b and pick the best value based on the performance on the validation set.

Singleton Mention Filter for OntoNotes Corpus. We employ the Illinois-Coref system (Chang *et al.*, 2012) to extract system mentions for our OntoNotes experiments, and observe that the number of predicted mentions is thrice the number of gold mentions. Since the training data

[3]http://sourceforge.net/p/lemur/wiki/RankLib/

provides the clustering supervision for only gold mentions, it is not clear how to train with the system mentions that are not part of gold mentions. A common way of dealing with this problem is to treat all the extra system mentions as singleton clusters (Durrett and Klein, 2013; Chang *et al.*, 2013). However, this solution most likely will not work with our current feature representation (i.e., NEW action is represented as a single indicator feature). Recall that to predict these extra system mentions as singleton clusters with our incremental clustering approach, the learned model should first predict a NEW action while processing these mentions to form a temporary singleton cluster, and then refrain from merging any of the subsequent mentions with that cluster so that it becomes a singleton cluster in the final clustering output. However, in OntoNotes corpus, the training data does not include singleton clusters for the gold mentions. Therefore, only the large number (57%) of system mentions that are not part of gold mentions will constitute the set of singleton clusters. This leads to a highly imbalanced learning problem because our model needs to learn (the weight of the single indicator feature) to predict NEW as the best action for a large set of mentions, which will bias our model to predict large number of NEW actions during testing. As a result, we will generate many singleton clusters, which will hurt the recall of the mention detection after post-processing. Therefore, we aim to learn a singleton mention filter that will be used as a pre-processor before training and testing to overcome this problem. We would like to point out that our filter is complementary to other solutions (e.g., employing features that can discriminate a given mention to be anaphoric or not in place of our single indicator feature, or using a customized loss to weight our ranking examples for cost-sensitive training)(Durrett and Klein, 2013).

Filter Learning. The singleton mention filter is a classifier that will label a given mention as "singleton" or not. We represent each mention m in a document by averaging the mention-pair features $\phi(m, m')$ of the k-most similar mentions (obtained by ranking all other mentions m' in the document with a learned ranking function R given m) and then learn a decision-tree classifier by optimizing the F1 loss. We learn the mention-ranking function R by optimizing the recall of positive pairs for a given k, and employ LambdaMART as our base ranker. The hyper-parameters are tuned based on the performance on the validation set.

3.5.2 Results

We first describe the results of the learned singleton mention filter, and then the performance of our Prune-and-Score approach with and without the filter. Next, we compare the results of our approach with several state-of-the-art approaches for coreference resolution.

Singleton Mention Filter Results. Table 3.1 shows the performance of the learned singleton mention filter with $k = 2$ noting that the results are robust for all values of $k \geq 2$. As we can see, the learned filter improves the precision of the mention detection with only small loss in the recall of gold mentions.

	Mention Detection Accuracy		
	P	R	F1
Before-filtering	43.18% (16664/38596)	86.99% (16664/19156)	57.71%
After-filtering	79.02% (15516/19640)	80.98% (15516/19156)	79.97%

Table 3.1: Performance of the singleton mention filter on the OntoNotes 5.0 development set. The numerators of the fractions in the brackets show the exact numbers of mentions that are matched with the gold mentions.

Prune-and-Score Results. Table 3.2 shows the performance of Prune-and-Score approach with and without the singleton mention filter. We can see that the results with filter are much better than the corresponding results without the filter. These results show that our approach can benefit from having a good singleton mention filter.

Filter settings	MUC	B^3	$CEAF_{\phi4}$	CoNLL
OntoNotes 5.0 Dev Set w. Predict Ment.				
O.S. (w.o. Filter)	66.73	53.40	44.23	54.79
P&S (w.o. Filter)	65.93	52.96	50.24	56.38
P&S (w. Filter)	**71.18**	**58.87**	**57.88**	**62.64**

Table 3.2: Performance of Prune-and-Score approach with and without the singleton mention filter, and Only-Score approach without the filter.

Table 3.3 shows the performance of different configurations of our Prune-and-Score approach. As we can see, Prune-and-Score gives better results than the configuration where we employ only the scoring function ($b = \infty$) for small values of b. The performance is clearly better than the degenerate case ($b = \infty$) over a wide range of b values, suggesting that it is not necessary to carefully tune the parameter b.

Comparison to State-of-the-Art. Table 3.4 shows the results of our **Prune-and-Score** approach

Pruning param. b	MUC	B^3	CEAF$_{\phi4}$	CoNLL
OntoNotes 5.0 Dev Set w. Predict Mentions				
2	69.12	56.80	56.30	60.74
3	70.50	57.89	57.24	61.88
4	71.00	58.65	57.41	62.35
5	**71.18**	**58.87**	**57.88**	**62.64**
6	70.93	58.66	57.85	62.48
8	70.12	58.13	57.37	61.87
10	70.24	58.34	56.27	61.61
20	67.97	57.73	56.63	60.78
∞	67.03	56.31	55.56	59.63

Table 3.3: Performance of Prune-and-Score approach with different values of the pruning parameter b. For $b = \infty$, Prune-and-Score becomes an Only-Scoring algorithm.

compared with the following state-of-the-art coreference resolution approaches: **HOTCoref** system (Björkelund and Kuhn, 2014); **Berkeley** system with the FINAL feature set (Durrett and Klein, 2013); **CPL^{3}M** system (Chang *et al.*, 2013); **Stanford** system (Lee *et al.*, 2013); **Easy-first** system (Stoyanov and Eisner, 2012); and **Fernandes et al., 2012** (Fernandes *et al.*, 2012). **Only Scoring** is the special case of our Prune-and-Score approach where we employ only the scoring function. This corresponds to existing incremental approaches (Daumé III, 2006; Rahman and Ng, 2011b). We report the best published results for CPL^{3}M system, Easy-first, and Fernandes et al., 2012. We ran the publicly available software to generate the results for Berkeley and Stanford systems with the updated CoNLL scorer. We include the results of Prune-and-Score for best b on the development set with singleton mention filter for the comparison. In Table 3.4, '-' indicates that we could not find published results for those cases. We see that results of the Prune-and-Score approach are comparable to or better than the state-of-the-art including Only-Scoring.

3.6 Summary

We introduced the Prune-and-Score approach for greedy coreference resolution whose main idea is to learn a pruning function along with a scoring function to effectively guide the search. We showed that our approach improves over the methods that only learn a scoring function, and gives comparable or better results than several state-of-the-art coreference resolution systems.

	MUC			B^3			$\text{CEAF}_{\phi 4}$			CoNLL
	P	R	F1	P	R	F1	P	R	F1	Avg-F1
a. Results on OntoNotes 5.0 Test Set with Predicted Mentions										
Prune-and-Score	81.03	66.16	**72.84**	66.90	51.10	57.94	68.75	44.34	53.91	61.56
Only-Scoring	75.95	61.53	67.98	63.94	47.37	54.42	58.54	49.76	53.79	58.73
HOTCoref	67.46	74.3	70.72	54.96	62.71	**58.58**	52.27	59.4	**55.61**	**61.63**
CPL^{3}M	-	-	69.48	-	-	57.44	-	-	53.07	60.00
Berkeley	74.89	67.17	70.82	64.26	53.09	58.14	58.12	52.67	55.27	61.41
Fernandes et al., 2012	75.91	65.83	70.51	65.19	51.55	57.58	57.28	50.82	53.86	60.65
Stanford	65.31	64.11	64.71	56.54	48.58	52.26	46.67	52.29	49.32	55.43
b. Results on OntoNotes 5.0 Test Set with Gold Mentions										
Prune-and-Score	88.10	85.85	86.96	76.82	76.16	76.49	80.90	74.06	**77.33**	**80.26**
Only-Scoring	86.96	84.52	85.73	74.51	74.25	74.38	79.04	70.67	74.62	78.24
CPL^{3}M	-	-	84.80	-	-	**78.74**	-	-	68.75	77.43
Berkeley	85.73	89.26	**87.46**	78.23	75.11	76.63	82.89	70.86	76.40	80.16
Stanford	89.94	78.17	83.64	81.75	68.95	74.81	73.97	61.20	66.98	75.14
c. Results on ACE2004 Culotta Test Set with Gold Mentions										
Prune-and-Score	85.57	72.68	**78.60**	90.09	77.02	**83.04**	74.64	86.02	**79.42**	**80.35**
Only-Scoring	82.75	69.25	75.40	88.54	74.22	80.75	73.69	85.22	78.58	78.24
CPL^{3}M	-	-	78.29	-	-	82.20	-	-	79.26	79.91
Stanford	82.91	69.90	75.85	89.14	74.05	80.90	75.67	77.45	76.55	77.77
d. Results on ACE2004 Newswire with Gold Mentions										
Prune-and-Score	89.72	75.72	**82.13**	90.89	76.15	**82.87**	72.43	86.83	**78.69**	**81.23**
Only-Scoring	86.92	76.49	81.37	88.10	75.83	81.51	73.15	84.31	78.05	80.31
Easy-first	-	-	80.1	-	-	81.8	-	-	-	-
Stanford	84.75	75.34	79.77	87.50	74.59	80.53	73.32	81.49	77.19	79.16
e. Results on MUC6 Test Set with Gold Mentions										
Prune-and-Score	89.53	82.75	86.01	86.48	76.18	**81.00**	60.74	80.33	**68.68**	**78.56**
Only-Scoring	86.77	80.96	83.76	81.72	72.99	77.11	57.56	75.38	64.91	75.26
Easy-first	-	-	**88.2**	-	-	77.5	-	-	-	-
Stanford	91.19	69.54	78.91	91.07	63.39	74.75	62.43	69.62	65.83	73.16

Table 3.4: Comparison of Prune-and-Score with state-of-the-art approaches. Metric values reflect version 7 of CoNLL scorer.

Our Prune-and-Score approach is a particular instantiation of the general idea of learning nearly-sound constraints for pruning, and leveraging the learned constraints to learn improved heuristic functions for guiding the search (See (Chen *et al.*, 2014) for another instantiation of this idea for multi-object tracking in videos). Therefore, other coreference resolution systems (Chang *et al.*, 2013; Durrett and Klein, 2013; Björkelund and Kuhn, 2014) can also benefit from this idea. One way to further improve the peformance of our approach is to perform a search in the Limited Discrepancy Search (LDS) space (Doppa *et al.*, 2014b) using the learned functions.

Chapter 4: Search-based Learning Algorithms for Multi-Task Structured Prediction

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 on 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. First, we establish the viability of search-based multi-task structured prediction for entity analysis by jointly solving named entity recognition, coreference resolution, and entity linking tasks on multiple benchmark datasets, namely ACE 2005 (NIST, 2005) and TAC-KBP 2015 (Ji *et al.*, 2015). 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.

4.1 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) (Lafferty *et al.*, 2001b), structured SVM (SSVM) (Tsochantaridis *et al.*, 2004), 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 (Daumé *et al.*, 2009; Ross *et al.*, 2011), heuristic functions (Daumé and Marcu, 2005b; Xu *et al.*, 2009), heuristic and cost functions (Doppa *et al.*, 2014a; Lam *et al.*, 2015), and coarse-to-fine knowledge (Weiss and Taskar, 2010).

There is some work on jointly solving two structured prediction tasks (Daumé and Marcu, 2005a; Finkel and Manning, 2009; Hajishirzi *et al.*, 2013; Li and Ji, 2014), but very little work on jointly learning and reasoning with three or more tasks (Singh *et al.*, 2013; Durrett and Klein, 2014). There are graphical modeling approaches that learn a global scoring function in the framework of CRFs or Structured SVMs (Finkel and Manning, 2009; Denis and Baldridge, 2007; Singh *et al.*, 2013; Durrett and Klein, 2014). Indeed, our work is inspired by the work of (Durrett and Klein, 2014) which employed graphical models and approximate inference via belief propagation for integrating multiple NLP subtasks for joint entity analysis. Integer linear programming (ILP) (Roth and Yih, 2005) formulation and inference is another potential approach, and has shown a lot of success in practice (Denis and Baldridge, 2007; Cheng and Roth, 2013). 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 (Daumé and Marcu, 2005b; Xu *et al.*, 2009; Daumé *et al.*, 2009; Doppa *et al.*, 2014a). Some approaches learn a scoring function to guide beam search in the space of *partial* structured outputs (incremental prediction approach) (Daumé and Marcu, 2005a; Bohnet and Nivre, 2012; Hatori *et al.*, 2012). In contrast, we perform search in the space of *complete* structured outputs (Doppa *et al.*, 2014b) 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.

4.2 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 (Pradhan *et al.*, 2012; Ji *et al.*, 2015; Ratinov and Roth, 2009). 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 co-reference 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 (Durrett and Klein, 2014).

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 $y_i^c \in \{1 \ldots i\}$ 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 $y_i^n \in \mathbf{T}$ is a semantic tag assigned to m_i ($\mathbf{T}$ is a constant set).
- Entity Linking decision y_i^l 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:

$$\Phi(x,y) = \underbrace{\Phi_1(x,y_1) \circ \cdots \circ \Phi_k(x,y_k)}_{\text{intra-task features}} \circ \underbrace{\cdots \circ \Phi_{(t_i,t_j)}(x,y_{t_i},y_{t_j}) \circ \cdots}_{\text{1st-order inter-task features}} \circ \underbrace{\cdots}_{\text{higher-order features}} \tag{4.1}$$

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.

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: $\Phi_{(c,n)}(x, y_c, y_n) \circ \Phi_{(c,l)}(x, y_c, y_l) \circ \Phi_{(n,l)}(x, y_n, y_l)$.

4.3 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.

[1] In some datasets, we also employ a slightly different definition $y_i^l = (q, e)$, 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_i^l 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.

4.3.1 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 (Kummerfeld *et al.*, 2015). 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.

Algorithm 4 Structured SVM Training

```
Input: $D$, training examples,
$\Phi(x, y)$, joint feature function
Output: $w$, the scoring function weights
 1: Initialization: active constraint set $A \leftarrow \emptyset$
 2: repeat
 3:     $w \leftarrow$ batch optimization with constraints $A$
 4:     for each training example $(x, y^*) \in D$ do
 5:         $\hat{y} \leftarrow$ BEAM-SEARCH-INFERENCE$(x, w)$
 6:         if $\hat{y} \neq y^*$ then
 7:             Add constraint $w \cdot \Phi(x, y^*) > w \cdot \Phi(x, \hat{y})$ to $A$
 8:         end if
 9:     end for
10: until convergence
11: return weights of the scoring function $w$
```

4.3.1.1 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.

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 (Yu and Joachims, 2009) 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.

4.3.1.2 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 Algorithm 5).

Algorithm 5 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 $\hat{y}$
```

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.

4.3.2 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 (Cohen and de Carvalho, 2005) 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.

4.3.3 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.

Algorithm 6 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 $P$
```

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 $\mathcal{R}$.

Algorithm 7 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 $P_{MAX}$
```

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.

4.3.4 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.

4.4 Experiments and Results

We evaluate our approach on two annotated datasets, ACE 2005 corpus (NIST, 2005) and TAC-KBP 2015 Entity Linking corpus (Ji *et al.*, 2015), 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.

4.4.1 Experimental Setup

Datasets. *ACE 2005* corpus contains 599 English documents. We follow the same setting as (Durrett and Klein, 2014) 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 (Bentivogli *et al.*, 2010) 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 (Ji *et al.*, 2015).

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 (Pradhan *et al.*, 2012). 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 (Ji *et al.*, 2015), 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 $CEAF_m$ (Luo, 2005). We also report the within document coreference scores using the same metrics as ACE 2005 (NIST, 2005). 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 (Lee *et al.*, 2011). 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 (Durrett and Klein, 2014). 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` (Chang *et al.*, 2015b) 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` (Chen and Guestrin, 2016) 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 $\mu : q \to E$ 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 $q \to e$ 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 (Wang *et al.*, 2014), we learn word, mention and entity embeddings by applying the skip-gram model (Mikolov *et al.*, 2013) 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 4.1 consists of the intra-task and inter-task features. This can be further decomposed as follows:

$$\begin{aligned}\Phi(x,y) &= \Phi_c(x,y_c) \circ \Phi_n(x,y_n) \circ \Phi_l(x,y_l) \circ \sum_i \phi_{(c,n)}(m_i,m_j,y_i^n,y_j^n) \circ \\ &\sum_i \phi_{(c,l)}(m_i,m_j,y_i^l,y_j^l) \circ \sum_i \phi_{(n,l)}(m_i,y_i^n,y_i^l), \text{ where } j = y_i^c \qquad (4.2)\end{aligned}$$

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 (Durrett and Klein, 2014) 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 `SharedRelatedWebsite`, `SameKBIDSameFreebaseType`, and `SameKBID` 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.

4.4.2 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 4.1 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.

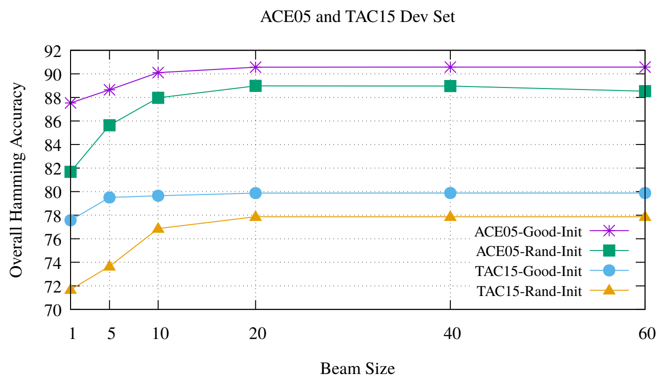


Figure 4.1: ACE 2005 and TAC-KBP 2015 Dev Set Performance with different beam sizes.

4.4.3 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 (Cohen and de Carvalho, 2005; Ross and Bagnell, 2010). 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.

	ACE 2005 Test			TAC-KBP 2015 Test			
Algorithms	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 4.1: ACE 2005 and TAC 2015 Test Set Performance with different pipeline orderings.

Table 4.1 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 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.

4.4.4 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.

Table 4.2.a shows the performance on ACE 2005 testing set for all 3 tasks. `Berkeley` (Durrett and Klein, 2014) 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

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 4.2: ACE 2005 Test Set Performance.

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.

Algm.	NER[2]	Link	NERLC	Within. Coref				Crs.Crf	Trn.
	Accu.	Accu.	Accu.	MUC	BCub	CEAF$_e$	CoNLL	CEAF$_m$	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 4.3: TAC-KBP 2015 Enitiy Linking Test Set Performance.

Table 4.3.a presents our results on TAC-KBP 2015. In this table, we add one more baseline `Rank-1st` (Ji *et al.*, 2015), 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 (Ji *et al.*, 2015). Again, our joint system outperforms

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

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.

4.4.5 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 4.2.b and 4.3.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 part (a), 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 4.4 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 4.5 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.	ACE05 Dev		TAC15 Dev	
α	Hamm.	Trn.Tm	Hamm.	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 4.4: 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.	ACE05 Dev		TAC15 Dev	
α	Hamm.	Tst.Tm	Hamm.	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 4.5: ACE 2005 and TAC 2015 Dev Set accuracy w.r.t. α with score-sensitive pruning.

4.4.6 Results for Cyclic Architecture

Table 4.2.c and 4.3.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.

	ACE 2005 Dev Set					
Cycle task	Unshared-Wt-Cyclic			Shared-Wt-Cyclic		
orderings	Coref.	NER	Link	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

Table 4.6: Unshared-Wt-Cyclic and Shared-Wt-Cyclic performance on ACE 2005 Dev w.r.t. task orderings.

`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 4.6, `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.

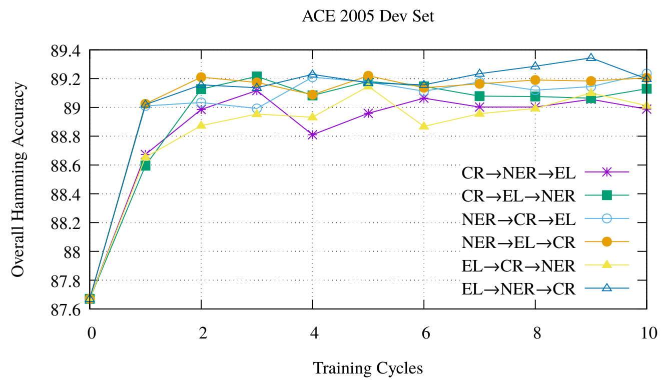
To verify our hypothesis, we present the ACE05 dev scores of the two algorithms in table 4.6. Each row corresponds to one task ordering. It is easy to observe that compared to `Unshared-Wt-Cyclic`, in `Shared-Wt-Cyclic` columns only the last tasks perform relatively well, while the first task can only 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.

In Figure 4.2 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.

4.5 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.



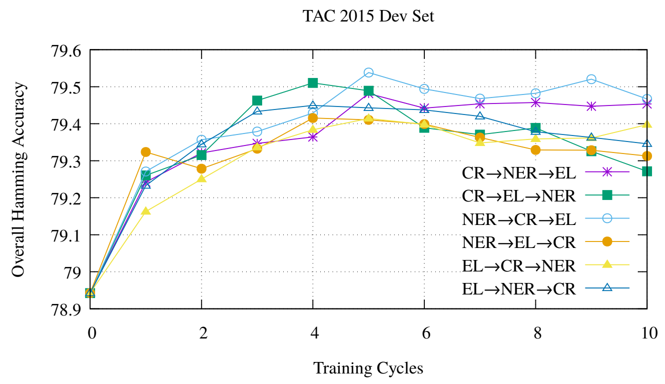


Figure 4.2: Hamming accuracy on ACE05 (up) and TAC15 (down) Dev set w.r.t. Unshred-Wt-Cyc training cycles.

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.

Chapter 5: $\mathcal{HC}$-Nets: A Framework for Search-based Deep Structured Prediction

The recent success of deep learning has proved the power of deep neural network models. A standard approach of structured prediction is to learn a scoring function $F(x, y)$ to score a candidate structured output y given a structured input x. The non-linear deep neural network model provides a stronger representation power for $F(x, y)$ than a linear function, and therefore makes it possible for the model to distinguish ground truth y^* from other outputs to achieve higher accuracy. Prior works (Belanger and McCallum, 2016; Belanger *et al.*, 2017; Gygli *et al.*, 2017; Tu and Gimpel, 2018) have shown that the incorporation of DNN models improves the state-of-the-art performance of structured prediction on a variety of application domains.

However, applying deep learning in structured prediction also faces some challenges. The first challenge is to ensure the stability and robustness of training and inference. Existing works use gradient-based inference to compute the $\arg\max$, which brings up several potential issues. To perform gradient-based inference, one needs to do relaxation over the output to continuous space, and then run gradient descent to get the prediction result. This procedure requires a very careful tuning of parameters, including the initial output, the step size, and the number of steps. Once the gradient descent was done, a rounding threshold is also required to cast the real valued output back to discrete space. These large number of interdependent parameters makes the gradient-based inference extremely difficult to implement and debug over different problem domains. Moreover, the relaxation of output to continuous space brings difficulties to enforce domain-specific constraints. (Lee *et al.*, 2019) relies on Lagrangian multiplier in the objective to perform the constrained optimization, which further adds to the complexity of implementation.

To deal with the challenges above, and leverage the advantages of both deep learning and search-based approaches in structure prediction, we combined the two to propose a new approach: *HC-Nets*. Referring to Table 2.2, we are trying to fill in the right-down cell with this approach. In $\mathcal{HC}$-Nets framework, we formulate structured prediction as a complete and discrete output space search problem. We choose complete output space instead of partial output space for several reasons. First, some loss functions are non-decomposable, and are only applicable when both the predicted and the ground truth complete outputs are available, e.g., example-based F1 score. A complete output would be essential for optimizing such a loss. Second, the partial output space does not have the anytime property, thus it cannot perform the coarse-to-fine inference given more time. Finally, in most applications, to control the branching factor (so that the search speed can be significantly improved), the partial output space explicitly defines an ordering of output predictions, which might limit the flexibility of the search. Moreover, the discrete output space makes the incorporation of prior knowledge constraints straightforward. Any output state that violates the hard constraints will be pruned off during the successor generation step in search.

The inference procedure of $\mathcal{HC}$-Nets consists of two stages: generation and selection. The first stage is generation step, where the greedy search is performed to uncover candidate outputs. The next stage is selection, where the best output is picked from the generated candidates. To guide the search, we employ two neural network functions: a heuristic network $\mathcal{H}$ for the generation stage and a cost network $\mathcal{C}$ for the selection stage. The decomposition of heuristic and cost functions makes the representation more expressive and learning more modular. To learn the functions $\mathcal{H}$ and $\mathcal{C}$, we developed a stage-wise training algorithm. Our learning of $\mathcal{H}$ is inspired by imitation learning, of which the target is to guide the search to execute the correct action at each search step, so that the search could uncover a set of high-quality outputs. The goal of $\mathcal{C}$ learning is similar to do ranking over the generated outputs, so that true best output can be chosen during the selection.

We evaluate our approach on three tasks: multi-label classification, handwriting recognition and semantic segmentation. Our experimental results show that $\mathcal{HC}$-Nets achieves better performance than prior methods on all three benchmark domains.

5.1 Related Work

The success of deep neural network models brings new opportunities to solving structured prediction problems. Structured prediction is especially important in deep learning because deep models are sample inefficient and structure is an effective way to constrain learning without using large amounts of training data (Liang *et al.*, 2008). Therefore, it is beneficial to integrate the advances in deep learning with structured prediction frameworks. Different instantiations of deep learning methods in SP incorporate ideas from both cost and search based learning approaches (Deshwal *et al.*, 2019).

Consider $\mathcal{C}(x, y)$ as a general cost (also referred as score or energy) function, where x is a structured input and y is a candidate structured output for x. $\mathcal{C}(x, y)$ typically consists of a combination of multiple local potential functions, i.e., functions defined over subsets of input/output variables. Classical deep networks allow us to learn complex unary features over structured input variables. In decoupled integration, we can employ the unary features learned from deep networks into existing structured prediction methods (Huang *et al.*, 2015).

To leverage structural dependencies, auto-regressive models including recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and attention networks (Sutskever *et al.*, 2014; Kim *et al.*, 2017) order the output variables and predict one variable at a time conditioned on the previous variables. This ordered sequence of output variables is typically generated using a decoding procedure based on greedy search or beam search. During the training phase, prediction of each variable is conditioned on the ground truth values of the previous output variables. As a result, the model is never exposed to its own error during training. This phenomenon is referred as *exposure bias* problem. To overcome this challenge, prior work has leveraged the general ideas from search-based methods. For example, LaSO approach is leveraged for beam search decoding (Wiseman and M. Rush, 2016) and advanced imitation learning methods are leveraged for greedy decoding (Bengio *et al.*, 2015). Another challenge with auto-regressive models is the mismatch between training loss (e.g., token-level cross entropy) and task loss defined over complete structure (e.g., BLEU score in translation). Reinforcement learning (RL) methods are employed with greedy decoding (Ranzato *et al.*, 2016) to alleviate this challenge by training over entire structure using the task loss as a reward function. Recent work presented a unified framework of training via both token-level maximum likelihood principle and RL (Tan *et al.*, 2019).

The two above-mentioned approaches employ a very simple form of structure among the output variables to learn representations: *no dependency structure between output variables* in decoupled integration and *linear ordering among output variables* in search-based learning. Therefore, these methods impose excessively strict inductive bias. To overcome these drawbacks, recent work explored tight integration of deep models with existing cost function learning approaches. Deep SP framework replaces clique potentials of conditional random fields with a deep network and approximates the partition function with loopy Belief Propagation (Chen *et al.*, 2015). Structured prediction energy networks (SPENs) framework allows us to learn a non-linear cost function over structured input-output pairs in the structured SVM training regime (Belanger *et al.*, 2017). SPENs fall within the general framework of energy-based learning (LeCun *et al.*, 2006). Deep value networks (DVNs) learn a non-linear regressor to approximate the negative loss value of a candidate structured output (Gygli *et al.*, 2017). Both SPENs and DVNs employs gradient-based inference in the relaxed continuous space. Approximate inference networks (InfNet) method (Tu and Gimpel, 2018) employs an additional trained deep neural network to directly produce the output of inference problem with a given cost function. A recent work developed a generalization of SPENs with improved results (Graber *et al.*, 2018) .

Some important challenges for methods that integrate deep models for structured prediction are as follows. *a) Incorporating constraints:* Many SP tasks require the structured outputs to satisfy some constraints (e.g., valid trees in parsing). Incorporating these constraints is a major challenge for methods that perform gradient-based inference and learning (Lee *et al.*, 2019). These constraints can be classified into three main categories: relational, logical, and scientific. Relational constraints enforce simple relations among entities which can be specified manually or mined from large amounts of unstructured text available on web. Logical constraints occur in domains where structured output variables are related by logical propositions (Xu *et al.*, 2018). Little attention has been paid to scientific constraints which require the predicted outputs to satisfy the true dynamics of our world based on physics (Stewart and Ermon, 2017). These constraints can also be thought of as prior knowledge for DL models and can improve their sample-efficiency. *b) Stability and Robustness:* The training of DL models in SP is prone to instability as discussed in earlier version of SPENs. Training of SPENs was improved (Belanger *et al.*, 2017), but this approach is prone to over-fitting. In our own experience, performance of DVNs is very sensitive to the parameters of gradient-based inference. Deep SP with non-linear output transformations approach (Graber *et al.*, 2018) doesn't support variable length structured outputs yet.

5.2 $\mathcal{HC}$-Nets Framework

5.2.1 $\mathcal{HC}$-Search Framework for Structured Prediction

$\mathcal{HC}$-Nets is an instantiation of $\mathcal{HC}$-Search (Doppa2014) with deep neural heuristic and cost functions. To apply the $\mathcal{HC}$-Search, one needs to formulate the original problem as a state-space search problem. Most important elements of this formulation are included in search space design. For example, how to represent an output as search state, how an action could change a state, and what constitute initial and terminal states. We introduce more details in the next subsection.

The inference in $\mathcal{HC}$-Nets is exactly the same as the search procedure of $\mathcal{HC}$-Search. Different from the other searched-based SP algorithms which employ only one scoring function, $\mathcal{HC}$-Search uses two learned functions: a heuristic function $\mathcal{H}$ and a cost function $\mathcal{C}$. The standard $\mathcal{HC}$-Search framework decomposes the structured prediction problem into three steps: 1) Find an initial complete output; 2) Explore a search tree of alternative candidate outputs rooted at the initial solution; and 3) Score each of these candidates to select the best one. Algorithm 8 combines these three steps in one search procedure.

Algorithm 8 Greedy Search Prediction with $\mathcal{HC}$-Nets

Input: x, structured examples input
$H(x, y)$, heuristic network; $C(x, y)$, cost network
$\langle Succ, I, T\rangle$, predefined search space, where I is initial state generator, T is terminal criteria and $Succ()$ is successor function

```
1: $B \leftarrow \emptyset$
2: $y_{last} \leftarrow \mathrm{I}(\mathbf{x})$ // compute initial state output
3: $y_{best} \leftarrow y_{last}$
4: **repeat**
5:     $B \leftarrow \textsc{Succ}(y_{last})$ // expand the current state
6:     $y_{last} \leftarrow \mathrm{argmin}_{y \in B}\, \mathcal{H}(x, y)$ // decide which state to expand in next step according to $H$
7:     $y_{best} \leftarrow \mathrm{argmin}_{y \in B}\, \mathcal{C}(x, y)$ // update the best output according to $C$
8: **until** terminal criteria $\mathrm{T}(x, y)$ was satisfied
9: **return** $y_{best}$
```

$\mathcal{HC}$-Search enjoys several advantages compared with other search-based approaches. First, in the standard approaches a global cost function must be trained to "defend against" the exponentially-large set of all wrong candidate outputs to the problem. This is expensive both computationally and in terms of sample complexity. It can require highly expressive representations (e.g., higher-order potentials). In contrast, the heuristic function $\mathcal{H}$ only needs to correctly rank the successors of each state that is expanded during the heuristic search in Step 2, and the cost function $\mathcal{C}$ only needs to correctly find the best of these in Step 3. These are much easier learning problems, and hence, simpler potential functions can be applied. Second, for making predictions there is no need to solve a global optimization problem at prediction time. In effect, the system learns not only how to score candidate solutions but also how to find good candidates - it learns to do inference more efficiently. Third, $\mathcal{HC}$-Search can be applied to non-decomposable loss functions. This is another advantage of using a search space formulation.

5.2.2 Search Space Design

Besides the search algorithm, a design of the search space is needed to formulate a structured prediction problem as state-space search. A search space explicitly defines all the key elements of search procedure. The search space consists of a tuple $\langle S, A, Succ, I, T\rangle$, where S denotes the search state. In most of applications, at the minimum a state contains an input-output pair (x, y). A and $Succ$ are respectively the definitions of action space and the successor function that controls the expansion of the search tree. I denotes initial state, which determines where the search tree starts, and T denotes the terminal state or terminal criteria, which decides when to stop the search. We introduce each elements in detail in the following.

Search State. A search state is a representation vector of a structured input-output pair, denoted by (x, y). It usually consists of two parts: one representation vector for the input x, and another for the output. Since we are searching a complete output space, the structured output would be fully labeled. We use y_i to denote the label of ith variable in y. Assume that $|y| = T$ and let $y_i \in V$, where V is a discrete set of all possible values y_i can take. In most problems, V is a constant set with a fixed size, say $|V| = N$. We construct the output vector in two steps. First, for each y_i we use a length N one-hot binary vector to represent y_i. For example, if $V = \{0, 1, 2, 3\}$, and $y_i = 2$, then the corresponding one-hot vector would be $(0, 0, 1, 0)$. Second, we concatenate the one-hot vectors of all y_i from $i = 1$ to $i = T$, and get a binary vector with length NT as the output representation. Concatenating the input and output vector, we got the representation of current search state.

Previous work has employed the gradient-based inference in structured prediction inference. Gradient based inference first relaxes the output representation from discrete space to continuous space. For instance, while originally $y_i \in \{0, 1\}$, we now allow $y_i \in [0, 1]$. Given an inference learning rate, the inference algorithm will run gradient ascent algorithm for a fixed number of steps using scoring function as a heuristic. During training, the relaxed output is directly used in computing the score and the weight update. In testing, this output will be rounded off and returned as the final result.

This inference approach raises a question: Is continuous or discrete representation better for neural network cost function learning? Intuitively, continuous representation provides richer input information for the network model. On the other hand, discrete space has its own advantages. First, in testing it does not need any rounding, which requires an additional rounding threshold parameter. Second, it is straightforward to define any hard constraints over the output. Finally, for a search-based approach, it is very easy to design actions to make changes over a discrete vector.

Action Space. In this work we will use the *Flipbit* search space since it is simple and straightforward to implement. In the standard flipbit search space, each action will pick only one variable in the output, and change its label from current one to an alternative value. We use the notation $(y_i \rightarrow z)$ to denote an action a, which means to change the i^{th} variable in y to a new label z. In order to add more flexibility to the search, we extend this action from "change only one variable" to "change at most k variables", and name it k-Flipbit search space. More formally, an action in k-Flipbit is a set of pairs rather than a single pair. $a' = (y_{i1} \rightarrow z_1), \cdots, (y_{ik} \rightarrow z_k)$, where $|a'| = k$. Each pair corresponds to one label change. Executing a' is equivalent to making changes on y according to each tuple in the set.

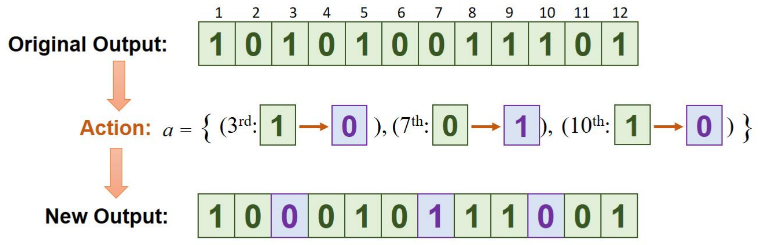


Figure 5.1: An example of action in k-Flipbit search space where $k = 3$.

Successor Function. Given a state, the successor function will generate new states by executing an action on the current state. To expand the search tree, the successor function will generate all possible actions for input state first, and then executes each action on current state to get a new state. Assume that current state is (x, y), and we want to choose k' labels in y to make changes. There are $\binom{T}{k'}$ possible ways to choose the k' labels. For each label there are $N - 1$ alternative values. So the total actions to make k' changes are $(N-1)^i \binom{T}{k'}$. Since we have $1 \leqslant k' \leqslant k$, the branching factor would be $\sum_{i=1}^{k} (N-1)^i \binom{T}{i}$. In order to avoid a large branching factor, in most problems we will limit $k \leqslant 3$.

Initial and Terminal States. Initial state is the starting point of the search. Complete output space requires the initial state having a fully labeled output as well. Due to the challenge of multiple local optima, the performance of search might be affected by the initial state. Therefore, in practice, one would need to carefully pick the initial state. In practice, to avoid overfitting to a particular initial state, some randomness should be added in the initial state generation. The simplest solution is using a purely random initial output.

Complete output space search usually does not have a hard criteria about what state should be the terminal state. Instead, a stopping criteria would be defined to decide when to stop search. For example, we can limit the search to a maximum search depth.

5.2.3 $\mathcal{H}$ and $\mathcal{C}$ Networks

The heuristic network $\mathcal{H}$ and cost network $\mathcal{C}$ are instantiations of scoring functions $F(x, y)$, which take a pair of structured input and output, and return a real value. x is feature vector of the input example, and y is the structured output vector. The value $\mathcal{H}(x, y)$ indicates how good the current state is to expand, or how far the current state is to the goal state (x, y^*). The value $\mathcal{C}(x, y)$ seeks to approximate $1 - l(x, y, y^*)$ without knowing y^*, where l is the output loss function, e.g., hamming error. The design of $\mathcal{H}$ and $\mathcal{C}$ networks is usually task-specific. For example, in multi-label classification problem, they could be 2-layer fully connected networks (FCNs). In image segmentation problem, the network might include three layer of CNNs and two layers of FCNs, etc. Note that our $\mathcal{HC}$-Nets framework does not have any requirements on the network architecture for both $\mathcal{H}$ and $\mathcal{C}$. There are, however, complex domain-specific trade-offs between the complexity of the networks and the efficiency of the resulting search with that architecture.

5.3 Heuristic and Cost Function Learning

5.3.1 Stage-wise Learning vs. Joint Learning

Since there are two functions in our approach, the learning needs to take their dependency into consideration. There are two possible general learning strategies based on the relations of the two functions.

Stage-wise Learning. Stage-wise learning strategy follows the original learning approach of $\mathcal{HC}$-Search. In the original $\mathcal{HC}$-Search learning, the error of the structured prediction learning is decomposed into two parts: 1) generation error, which occurred when heuristic function fails to guide the search uncovering a search tree with ground truth outputs; and 2) selection error, which is caused by the cost function when it fails to choose the "best output" (according to output loss function l) in the generated candidates. Thus, the training contains two separate stages: heuristic function training first, followed by the cost function training conditioned on the learned heuristic function. Stage-wise learning is easy to implement in practice, but is imperfect in some respects. For example, cost function training depends on an existing heuristic function, but can never influence the heuristic training. In other words, this learning strategy does not follow the end-to-end learning principle.

Joint Learning. Another strategy is to learn the two functions jointly. Under this formulation, the error of the heuristic is a function of the cost, and similarly the error of the cost is a function of the heuristic. The training algorithm will iteratively update the heuristic and the cost function one after another. When we update one function, the other one would be treated as constant.

The current thesis only evaluates the stage-wise learning approach. The formulation and experimental evaluation of joint learning will be an important part of future work.

5.3.1.1 Heuristic Function Learning

The key idea of the heuristic function training is inspired by imitation learning. The target is to guide the search to execute the correct action at each search step. To define the term "correct action," we need to have an output loss function $l(x, \hat{y}, y^*)$, given input x and the ground truth output y^*. In our search algorithm, the "correct action" is defined as the best action whose output has the minimum l among all its siblings. In another words, if y_{t+1} is the output of y_t by executing action a_t at step t, then a_t is the best action if $l(x, \hat{y_{t+1}}, y^*)$ is the minimum among all other candidate successors.

In heuristic function training stage, we run greedy search to expand a search tree for each example. This search procedure also needs a scoring function to guide. There are two possible choices: using l directly, or using the current learned heuristic function $\mathcal{H}$, named *on-trajectory* and *off-trajectory*, respectively. Since we employ the "update after each mini-batch" strategy, at each search step we only aggregate training data for the current step instead of directly doing the weight update. The update would be postponed until the all examples of each mini-batch have been processed. Similar to most machine learning libraries, we reduce the update procedure as a blackbox optimization problem, and provide two possible reductions for the blackbox optimizer: *regression-based* and *ranking-based* reductions.

Blackbox Optimizer. A blackbox optimizer usually takes feature vectors from a set of input-output pairs, and a loss function defined over these pairs as the objective. These outputs are a subset of uncovered outputs in the search tree. It also takes some learning parameters, and optimises the model with respect to the objective.

Training Data Aggregation. The training data aggregation step collects a set of example outputs as the input to the optimizer. Ideally, the optimizer should take the outputs in the entire search tree of each example as input, but for some tasks this would cause scalability issue due to the large size of the search trees. Another alternative choice is only including the last output on the search path, because the last output is the best output on the trajectory. In this work, we apply `TrajOutput`, which includes outputs that were expanded during search (trajectory outputs), because these outputs are the "best outputs" among all their siblings, and it is a reasonable trade-off between the diversity and scalability of outputs in the set.

5.3.1.2 Cost Function Learning

The cost function learning is relatively simple. In cost function learning, the generation stage can be viewed as a blackbox. This blackbox outputs a finite set of candidate outputs, which correspond to the search tree guided by heuristic function. At this point, the $\operatorname{argmin}_y \mathcal{C}(x, y)$ computation is not intractable anymore since y is chosen from the finite candidate set rather than from the entire solution space. If we treat each output in this set as a "candidate label" and the best output in terms of l as the "correct label", then cost function learning can be easily reduced to a rank learning problem.

Algorithm 9 is the general function learning algorithm of $\mathcal{HC}$-Nets. The heuristic and cost function learning are incorporated in a single algorithm. We will discuss the optimization in the following sections.

5.3.2 Reduction to Regression Learning

The goal of training is to learn a function f such that the searcher will behave exactly the same as it was guided by the loss function l. Thus,if f and l can return the same value for any given input and predicted output, the goal would be achieved. That is the motivation for regression-based learning. Formally, we define the regression loss over all input-output pairs in the aggregate data for a given function as

$$\sum_{(x,y,y^*)\in R} L(x, y, y^*, l, f) = \sum_{(x,y,y^*)\in R} l(x, y, y^*) \log f(x, y) - (1 - l(x, y, y^*)) \log(1 - f(x, y)) \tag{5.1}$$

Algorithm 9 Heuristic and Cost Function Training

```
Input: $D = \{\mathbf{x}, \mathbf{y}^*\}^N$, structured input-output training examples
$optimizer$, blackbox optimizer to do weight update
$b = 1$, set beam size as 1 to apply greedy search;  $m$, mini-batch size
Output: $\theta_H$, $\theta_C$, network weights of heuristic and cost function
 1: $R \leftarrow \emptyset$
 2: for $epoch \leftarrow 0$ to $MAX\_EPOC$ do
 3:     for $(\mathbf{x}, \mathbf{y}^*) \leftarrow D$ do
 4:         $y_0 \leftarrow \textsc{InitialOutput}(\mathbf{x})$
 5:         $B \leftarrow \{y_0\}$
 6:         repeat
 7:             $y_{best} \leftarrow \textsc{PopBest}(B; l)$ // change $l$ to $\theta$ to do off-trajectory learning
 8:             $B \leftarrow B \cup \textsc{Successor}(y_{best})$ // expand the current state
 9:             $R \leftarrow R \cup \{y_{best}\}$ // add trajectory output into aggregation set
10:             if $y_{best}$ is local optimal then
11:                 break
12:             end if
13:             $B \leftarrow$ keep top-$b$ states of $B$
14:         until maximum search steps
15:         if  end of mini-batch size $m$ then
16:             $\theta_H \leftarrow optimizer(R, \theta_H, l)$ // update heuristic weights
17:             $\theta_C \leftarrow optimizer(R, \theta_C, l)$ // update cost weights
18:             $R \leftarrow \emptyset$ // clear cached training data
19:         end if
20:     end for
21: end for
22: return $\theta_H$, $\theta_C$
```

which is the cross-entropy loss over f and l. We can use stochastic gradient descent to do the optimization as in Algorithm 10.

5.3.3 Reduction to Rank Learning

Although regression-based optimizer is easy to understand and straightforward to implement, it also has some drawbacks. First, it learns more than what we really need. Note that in our search, to pick the best successor, we only need to ensure that the best state would rank higher than others, but do not care about the exact value of the heuristic or cost. Second, regression based

Algorithm 10 Regression-based Optimizer

```
Input: $R = \{\mathbf{x}, \mathbf{y}^*\}^M$, aggregated input-output training examples
$l(y, y^*)$, output true loss function
$\theta$, the weights before update
1: repeat
2:     for $(\mathbf{x}, \mathbf{y}, \mathbf{y}^*) \leftarrow R$ do
3:         $l^* \leftarrow l(\mathbf{y}, \mathbf{y}^*)$
4:         $L \leftarrow l^* \log f(\mathbf{x}, \mathbf{y}; \theta) - (1 - l^*) \log(1 - f(\mathbf{x}, \mathbf{y}; \theta))$
5:         $\theta \leftarrow \theta - \frac{d}{d\theta} L$
6:     end for
7: until maximum iteration or convergence
8: return $\theta$
```

learner does not take the internal relations of each expanded sibling states (state ranking list) into consideration, but treats each state as an independent regression point. The information in the search procedure is not fully exploited.

A ranking-based reduction can overcome these potential issues. Our rank learner will try to minimize the ranking loss. The ranking loss is defined by counting the number of violated pairs in a ranking list. More specifically, given an input x, for any pair of predicted outputs y_1 and y_2 in the same ranking list, pair $\langle y_1, y_2 \rangle$ is a violated pair if $(f(x, y_1) - f(x, y_2))(l(x, y_1, y^*) - l(x, y_2, y^*)) < 0$. Instead of considering all possible pairs, we only include pairs that involve the true best output $\tilde{y}$ at current step, which is the output with the minimum l value in the ranking list. Formally, we first define $\tilde{y} = \arg\min_{y \in B} l(x, y, y^*)$, and then define the ranking loss over each element in the ranking list B as:

$$\sum_{y \in B \backslash \tilde{y}} L_{rk}(x, y, \tilde{y}, y^*, l, f) = \sum_{y \in B \backslash \tilde{y}} \max \big\{0, -\big(f(x, \tilde{y}) - f(x, y)\big)\big(l(x, \tilde{y}, y^*) - l(x, y, y^*)\big)\big\} \tag{5.2}$$

5.4 Experiments and Results

5.4.1 Experimental Setup

We evaluate our frameworks on three tasks. We first choose multi-label classification since it is clearly defined and is a relatively simple problem with standard benchmark datasets and plenty

Algorithm 11 Ranking-based Optimizer

```
Input: $R = B^M$, where $B = \{\mathbf{x}, \mathbf{y}^*\}^L$, aggregated input-output training examples
$l(y, y^*)$, output true loss function
$\theta(x, y)$, the model before update
1: repeat
2:     for $B \leftarrow R$ do
3:         $\tilde{y} = \arg\min_{y \in B} l(x, y, y^*)$
4:         $L = \sum_{y \in B \setminus \tilde{y}} \max\left\{0, -\big(f(x, \tilde{y}) - f(x, y)\big)\big(l(x, \tilde{y}, y^*) - l(x, y, y^*)\big)\right\}$
5:         $\theta \leftarrow \theta - \frac{d}{d\theta} L$
6:     end for
7: until maximum iteration or convergence
8: return $\theta$
```

of baseline performances. Then, we apply our approach on real world problems in a variety of domains, including hand-written word recognition and semantic image segmentation.

Multi-label Classification. Multi-label classification task is to predict a binary vector for an input example, where each bit indicates whether a corresponding label should be given to the input. Each input example is represented with a feature vector, and can be assigned to one or more possible labels.

We evaluate our approach on three datasets: Yeast, Bibtex and Bookmarks. Following the setting of the prior work, we pick the Bibtex and Bookmarks datasets in order to compare against SPEN(Belanger *et al.*, 2017), DVN(Gygli *et al.*, 2017) and InfNet(Tu and Gimpel, 2018). To verify the performance on the relatively small datasets in terms of training set and label size, we add an additional dataset Yeast. Yeast has 14 candidate labels, while Bibtex and Bookmarks contain 158 and 208 labels respectively. We apply the standard train/test split of 1500/917 for Yeast, 4800/2515 for Bibtex and 60000/27856 for Bookmarks. The macro-averaged F1 accuracy is used as the evaluation metric.

Our heuristic and cost function network for multi-label classification is derived from (Belanger and McCallum, 2016; Gygli *et al.*, 2017). The following figure 5.2 shows the structure of the neural network. In our experiments, we set the maximum number of epochs to 300. For each dataset we randomly split out 5% of the examples as validation set. We use the learning rate of 0.005 for Yeast, and 0.1 for the other two datasets.

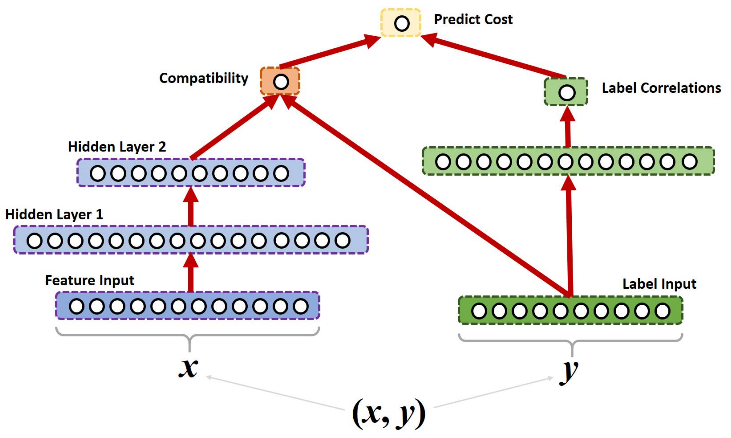


Figure 5.2: The heuristic and cost function network in multi-label classification.

Word Recognition Word recognition is the task of recognizing five-letter English words from given images. It is tested on a synthetic dataset constructed by repeatedly choosing from a list of 50 common English words, and corresponding letter images from the Chars74K dataset.The image of each word is cropped into five letters, and each of them is rendered as a 28x28 pixel image. Following the setting of (Graber *et al.*, 2018), the training, validation, and test sets for these experiments consist of 1000, 200 and 200 examples, respectively. We evaluate the performance with character-wise accuracy and word-wise accuracy.

Our network for this task contains two parts. The first part measures the consistency between the letter image to a letter label, which starts from the output of AlexNet(Krizhevsky *et al.*, 2012), and is followed by two fully connected layers. The second part, consists of a two-layer feed forward networks, and captures the relevance between labels values (similar to the pairwise, ternary, quadery potentials in CRFs). A linear weighted sum of two parts, followed by a sigmoid activation are the last two layers before the output.

Image Segmentation Semantic image segmentation task is to segment the give image into different regions, and label each region according to their meaning. For example, if the image contains a horse on the grassland, then the system should label the pixels that cover the horse as `HORSE`, and the other region as `GRASS`. To simplify the task, we will only focus on the foreground-background segmentation which contains only two classes. We choose the Weizmann Horses database, consisting of 328 images of horses, including the corresponding ground truth masks. Following the setup of DVN, we use the train/validation/test splits of 196/66/66 images, respectively. All the images and masks are re-scaled to 32x32 pixels. The results will be evaluated with intersection over union (IoU) to ensure that it is comparable to the baselines.

We will follow the same design of network architecture as DVN, shown in Figure 5.3. The network itself contains three convolution layers followed by two fully connected layers.

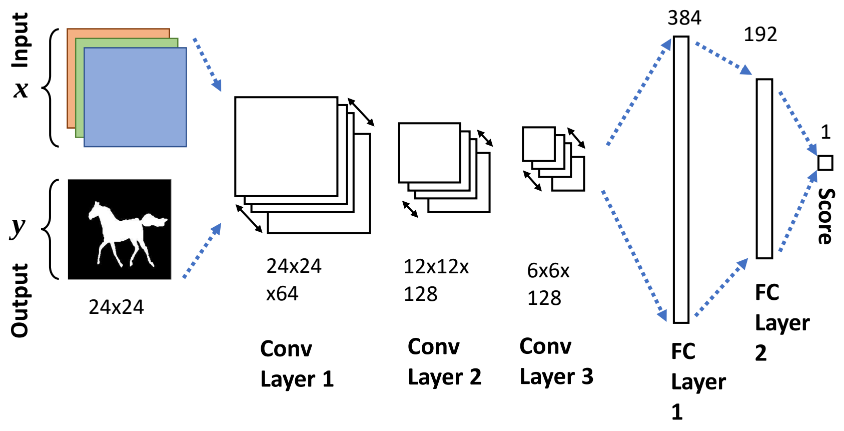


Figure 5.3: The network architecture for image segmentation.

5.4.2 Results and Discussion

We compare our performance with state of the art approaches, including SPEN(Belanger *et al.*, 2017), DVN(Gygli *et al.*, 2017), InfNet(Tu and Gimpel, 2018), NLStruct(Graber *et al.*, 2018). In order to compare the effects of initialization methods, we also present the results with pure random initialization and with learned classifier output as initialization. In Table 5.1, `⋆-Rand-Init` denotes the result with random initialization and `⋆-Learned-Init` is the result with an initialization predicted by an i.i.d. classifier for each output label. `Regr` and `Rank` represent the regression and ranking reduction respectively.

Comparison to State of the Art From the results in Table 5.1, we can see that our approach gets a comparable or better accuracy on most the datasets. The only exception is the ranking reduction performance on Horses dataset. This is because our currently offline rank learner only accept the simple feed-forward network architecture. We believe the performance can be further

	Multi-Label			**Word Recog.**	**Image Segm.**
Algs.	Yeast	Bibtex	Bookmarks	HW-Words	Horse32x32
	Averaged F-1 Acc.			Char Acc.	IoU
SPEN	63.8	38.1	33.9	42.26	75.45
DVN	63.8	44.7	37.1	-	84.00
InfNet	-	42.2	37.6	37.95	69.31
NLStruct	-	-	-	44.37	81.86
Regr-Rand-Init	62.5	43.2	34.9	39.02	74.62
Regr-Learned-Init	62.6	44.7	37.8	45.14	82.55
Rank-Rand-Init	62.6	43.8	36.4	45.52	68.8
Rank-Learned-Init	64.0	45.9	38.7	45.98	73.2

Table 5.1: Accuracy comparison with the SOTA approaches.

improved with a more flexible rank l earning algorithm. Our approaches can achieve a larger performance margin against the baselines on Bibtex and Bookmarks. This is partly because these two datasets are relatively large, and can provide more training materials during search, while the other three datasets contain less than 1000 examples. Our program did not outperform DVNs on Horses dataset due to several reasons. The main reason is the much bigger structure size of Horse images compared to the other two tasks, which poses at least two challenges. First, the search with a large output size is more sensitive to the initial output. Second, the large size of the output usually requires more steps to reach the target state in generation step, which would lead to a longer trajectory and more generated outputs. To control the complexity of heuristic search stage, we did not directly apply search to the initial output, but sampled a proportion of variables and assigned them with initial output values while keeping the others at ground truth values. This mismatch between training and testing might have also hurt the performance. Considering the initialization methods, the `Learned-Init` performs better than `Rand-Init` in most cases. This is not surprising because the `Learned-Init` provides a better starting point of search that improves the quality of generated outputs in limited number of search steps. The comparison between the rows of `Regr-*` and `Rank-*` shows that ranking reduction is more effective learning reduction compared to regression, which reflects the drawbacks of regression-based reduction we mentioned at the beginning of Section 5.3.3.

Ablation Study of Generation and Selection Errors Table 5.2 shows another analysis on varying the maximum search depth of generation stage. We present both generation and selection accuracy with two different initialization methods. The generation accuracy is the result of running `HL-Search`, which means that we apply our learned $\mathcal{H}$ function in the generation stage, but use the output loss function l as the cost function in the selection stage, therefore we can decompose the error of $\mathcal{H}$ and the error of $\mathcal{C}$.

Depth	Gen. Acc.		Real Acc.	
	Rand-Init	Learned-Init	Rand-Init	Learned-Init
Yeast				
2	0.531	0.674	0.462	0.579
5	0.755	0.816	0.553	0.623
10	0.816	0.831	0.564	0.627
14	too slow	too slow	0.598	0.629
Bibtex				
2	0.698	0.722	0.312	0.375
3	0.722	0.748	0.384	0.421
4	0.759	0.801	0.385	0.425
5	0.762	0.811	0.385	0.426
Bookmarks				
2	0.791	0.84	0.285	0.344
3	0.791	0.856	0.279	0.357
4	0.792	0.882	0.292	0.358
Words Recognition				
1	0.22	0.41	0.15	0.28
5	0.56	0.67	0.32	0.35
10	0.74	0.88	0.38	0.41
15	0.83	0.91	0.37	0.404
Horse32x32				
10	0.24	0.68	0.164	0.531
20	0.57	0.73	0.415	0.628
50	0.79	0.86	0.614	0.719
65	0.88	0.93	0.628	0.698

Table 5.2: Accuracy with different generation search depths.

Although the generation accuracy keeps increasing as the depth goes deeper as expected, the selection accuracy saturates at some depth. This is because it becomes increasingly challenging to pick the right output from a large number of generated outputs. Second, the `Learned-Init` performs better then `Rand-Init` on both generation and selection accuracy with a big margin when the depth limit is small, but the gaps vanish at larger depths.

5.5 Summary

In this work, we proposed the $\mathcal{HC}$-Nets framework to solve the structured prediction problem with a two-stage search approach. In generation stage, we run greedy search under the k-flipbit search space to uncover a set of candidate outputs, guided by the heuristic function. In selection stage, the cost function is employed to find the best prediction among the generated output set. The heuristic and cost function are neural network models. We provide two reductions of learning algorithms: the regression-based reduction and the ranking-based reduction. Our results shows that $\mathcal{HC}$-Nets outperform the baselines over most presented domains, which proves the effectiveness of our approach. We believe the performance ranking reduction learning can be further improved with a more flexible rank learning algorithm.

Chapter 6: Conclusions and Future Work

In this thesis, we extended the search-based approaches for structured prediction in several new directions. This section summarizes the main contributions and outlines future research directions.

6.1 Contributions of the Thesis

We developed a learning approach called *Prune-and-Score* to improve the accuracy of greedy structured prediction for search spaces with large action spaces. The key idea is to learn two functions: a pruning function that prunes bad decisions and a scoring function that then selects the best remaining decision. We reduce the problem of learning these two functions to rank learning, which allows us to leverage powerful and efficient off-the-shelf rank learners. We apply this method to the problem of co-reference resolution in natural language processing. The evaluation results on OntoNotes dataset proves that it is competitive with the state-of-the-art approaches at the time of its development and compares favorably with a greedy search-based approach that uses a single scoring function.

To address the SP problems that involve multiple tasks, we explored search-based methods to solve *multi-task structured prediction* (MTSP), in the context of a specific NLP application: entity analysis. We employed complete output space best first beam search as the inference method, and learned linear scoring functions with structural SVM. Due to the large output size with multiple tasks, the inference efficiency becomes critical and challenging. We studied three search architectures, the widely used "pipe line" and "joint" architectures, and a novel "*cyclic*" architecture, which exhibits the best trade-off between speed and accuracy of training and inference. 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 performed empirical evaluation of the proposed architectures for entity analysis, and achieved the state-of-the-art results on two datasets.

We also extended the original $\mathcal{HC}$-Search (Doppa *et al.*, 2013b, 2014a) framework to deep neural networks and created a new framework called $\mathcal{HC}$-Nets. The motivation for this work is to automate the human engineering of features in structured prediction and overcome the implementation issues in the existing deep structured prediction methods including the hardness of incorporating constraints, the instability of training and the difficulty of debugging. To deal with these issues, we formulated structured prediction as a complete and discrete output space search problem, and decomposed the inference procedure into two stages: generation and selection. We employed two neural network functions to guide the search – a heuristic function $\mathcal{H}$ for generating candidate outputs and a cost function $\mathcal{C}$ for selecting the best output from the generated candidates. The discrete output space makes the incorporation of prior knowledge constraints straightforward, and the separation of $\mathcal{H}$ and $\mathcal{C}$ functions makes the representation more expressive and learning more modular. The evaluation on three benchmark tasks shows that our approach can achieve competitive performance.

6.2 Future Work

We now list some important open problems and future directions for this line of research.

- **End-to-End MTSP with Deep Neural Networks.** In Chapter 4, we presented a solution for solving the multi-task structured prediction with beam search guided by linear cost functions. In recent work on deep neural networks, end-to-end learning with large scale training data has become a promising direction to improve the accuracy of the state of the art NLP systems on a variety of tasks. The recent progress on BERT (Devlin *et al.*, 2018) model has partially proved this point. It would be an interesting topic to explore how to integrate this idea with search-based MTSP approaches. More specifically, this research would try to answer the questions like the following: how to formulate search-based SP approach in an end-to-end learning framework; what tasks can be solved jointly; and how to make trade-offs between the efficiency and model complexity in both training and testing of deep neural networks.
- **$\mathcal{HC}$-Nets Learning for Variable Length Sequences.** In $\mathcal{HC}$-Nets framework, to ensure that scored values of different input-output pairs are comparable, the network output values should lie in the same range. However, in many structured prediction applications, the input and output examples are both sequences of variable length. Because the inputs of neural networks are of fixed dimension, creating a fixed length representation for these non-fixed length inputs is a challenge. Currently, there are two possible candidate directions for this problem. The first is to use the last hidden state vector of recurrent models like LSTMs or GRUs. The second is to apply one dimensional CNNs with the fixed span filters with shared parameters over the input. While SP-methods based on partial output space explored both these directions, they have not

been explored for complete output space search to the best of our knowledge.

- **Joint Learning of $\mathcal{H}$ and $\mathcal{C}$ networks in $\mathcal{HC}$-Nets.** In our current proposed $\mathcal{HC}$-Nets framework, we formulate the learning of $\mathcal{H}$ and $\mathcal{C}$ functions as stage-wise learning problem. The $\mathcal{H}$ learning is done first, and then conditioned on learned $\mathcal{H}$, we train $\mathcal{C}$-learning. Although such a formulation of $\mathcal{H}$ and $\mathcal{C}$ learning works well, it is not fully satisfactory because it does not follow the end-to-end learning philosophy of deep learning. In particular, during the cost function learning, the update of $\mathcal{C}$ would never be able to backpropagate to $\mathcal{H}$. The success of inference network (Tu and Gimpel, 2018) provides an inspiration for solving this problem and seems extendable to $\mathcal{HC}$-Nets. The big picture of this idea is to formulate the learning in an analogy to generative adversarial network (GAN), where $\mathcal{H}$ performs the role of the generator network, and $\mathcal{C}$ the discriminator network.

Bibliography

Amit Bagga and Breck Baldwin. Algorithms for Scoring Coreference Chains. In *Proceedings of the International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, 1998.

David Belanger and Andrew McCallum. Structured Prediction Energy Networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 983–992, 2016.

David Belanger, Bishan Yang, and Andrew McCallum. End-to-End Learning for Structured Prediction Energy Networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 429–439, 2017.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2015.

Eric Bengtson and Dan Roth. Understanding the Value of Features for Coreference Resolution. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 294–303, 2008.

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 *Proceedings of the 2nd Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, 2010.

Anders Björkelund and Jonas Kuhn. Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 47–57, 2014.

Bernd Bohnet and Joakim Nivre. A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, 2012.

Christopher Burges. From RankNet to LambdaRank to LambdaMART: An Overview. *Microsoft Technical Report*, 2010.

Kai-Wei Chang, Rajhans Samdani, Alla Rozovskaya, Mark Sammons, and Dan Roth. Illinois-Coref: The UI system in the CoNLL-2012 shared task. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 113–117, 2012.

Kai-Wei Chang, Rajhans Samdani, and Dan Roth. A Constrained Latent Variable Model for Coreference Resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 601–612, 2013.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to Search Better than Your Teacher. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 2058–2066, 2015.

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.

Kai-Wei Chang, Shyam Upadhyay, Gourab Kundu, and Dan Roth. Structural Learning with Amortized Inference. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Sheng Chen, Alan Fern, and Sinisa Todorovic. Multi-Object Tracking via Constrained Sequential Labeling. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Liang-Chieh Chen, Alexander G. Schwing, Alan L. Yuille, and Raquel Urtasun. Learning Deep Structured Models. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1785–1794, 2015.

Xiao Cheng and Dan Roth. Relational Inference for Wikification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

William W. Cohen and Vitor Rocha de Carvalho. Stacked Sequential Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

Michael Collins and Brian Roark. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, 2004.

Michael Collins. Discriminative Reranking for Natural Language Parsing. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 175–182, 2000.

Aron Culotta, Michael L. Wick, and Andrew McCallum. First-Order Probabilistic Models for Coreference Resolution. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pages 81–88, 2007.

Hal Daumé, III and Daniel Marcu. A Large-scale Exploration of Effective Global Features for a Joint Entity Detection and Tracking Model. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

Hal Daumé, III and Daniel Marcu. Learning As Search Optimization: Approximate Large Margin Methods for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 169–176, 2005.

Hal Daumé, III, John Langford, and Daniel Marcu. Search-based Structured Prediction. *Journal of Machine Learning Research - Proceedings Track*, 2009.

Hal Daumé III. *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, 2006.

Pascal Denis and Jason Baldridge. Joint Determination of Anaphoricity and Coreference Resolution using Integer Programming. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2007.

Pascal Denis and Jason Baldridge. Specialized Models and Ranking for Coreference Resolution. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pages 660–669, 2008.

Aryan Deshwal, Janardhan Rao Doppa, and Dan Roth. Learning and Inference for Structured Prediction: A Unifying Perspective. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Output Space Search for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2012.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: Learning Heuristics and Cost Functions for Structured Prediction. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: Learning Heuristics and Cost Functions for Structured Prediction. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A Learning Framework for Search-based Structured Prediction. *Journal of Artificial Intelligence Research (JAIR)*, 49, 2014.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured Prediction via Output Space Search. *Journal of Machine Learning Research (JMLR)*, 15:1317–1350, 2014.

Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. HC-Search for Multi-Label Prediction: An Empirical Study. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

Janardhan Rao Doppa. Integrating Learning and Search for Structured Prediction. *Ph.D. Thesis, Oregon State University*, 2014.

Greg Durrett and Dan Klein. Easy Victories and Uphill Battles in Coreference Resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1971–1982, 2013.

Greg Durrett and Dan Klein. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. In *Transactions of the Association for Computational Linguistics*, 2014.

Greg Durrett, David Leo Wright Hall, and Dan Klein. Decentralized Entity-Level Modeling for Coreference Resolution. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 114–124, 2013.

Pedro F. Felzenszwalb and David A. McAllester. The Generalized A* Architecture. *Journal of Artificial Intelligence Research (JAIR)*, 29:153–190, 2007.

Eraldo Rezende Fernandes, Cícero Nogueira dos Santos, and Ruy Luiz Milidiú. Latent Structure Perceptron with Feature Induction for Unrestricted Coreference Resolution. In *Proceedings of International Conference on Computational Natural Language Learning (CoNLL)*, pages 41–48, 2012.

Jenny Rose Finkel and Christopher D. Manning. Joint Parsing and Named Entity Recognition. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2009.

Thomas Finley and Thorsten Joachims. Supervised Clustering with Support Vector Machines. In *Proceedings of International Conference on Machine Learning (ICML)*, 2005.

Yoav Goldberg and Joakim Nivre. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414, 2013.

Colin Graber, Ofer Meshi, and Alexander Schwing. Deep Structured Prediction with Nonlinear Output Transformations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 6320–6331. Curran Associates, Inc., 2018.

Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep Value Networks Learn to Evaluate and Iteratively Refine Structured Outputs. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1341–1351, 2017.

Aria Haghighi and Dan Klein. Coreference Resolution in a Modular, Entity-Centered Model. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, 2010.

Hannaneh Hajishirzi, Leila Zilles, Daniel S. Weld, and Luke S. Zettlemoyer. Joint Coreference Resolution and Named-Entity Linking with Multi-Pass Sieves. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 289–299, 2013.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based Structured Prediction. *Machine Learning Journal (MLJ)*, 75(3):297–325, 2009.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, 2012.

Liang Huang, Suphan Fayong, and Yang Guo. Structured Perceptron with Inexact Search. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, 2012.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv*, abs/1508.01991, 2015.

Heng Ji, Joel Nothman, Ben Hachey, and Radu Florian. Overview of TAC-KBP2015 Tri-lingual Entity Discovery and Linking, 2015.

Roni Khardon. Learning to Take Actions. *Machine Learning Journal (MLJ)*, 35(1):57–90, 1999.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured Attention Networks. *arXiv*, abs/1702.00887, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 1097–1105, 2012.

Jonathan K. Kummerfeld, Taylor Berg-Kirkpatrick, and Dan Klein. An Empirical Analysis of Optimization for Max-Margin NLP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of International Conference on Machine Learning (ICML)*, 2001.

Michael Lam, Janardhan Rao Doppa, Xu Hu, Sinisa Todorovic, Thomas Dietterich, Abigail Reft, and Marymegan Daly. Learning to Detect Basal Tubules of Nematocysts in SEM Images. In *ICCV Workshop on Computer Vision for Accelerated Biosciences (CVAB)*. IEEE, 2013.

Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G. Dietterich. HC-Search for Structured Prediction in Computer Vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4923–4932, 2015.

Yann LeCun, Sumit Chopra, Raia Hadsell, Fu Jie Huang, and et al. A Tutorial on Energy-based Learning. In *PREDICTING STRUCTURED DATA*. MIT Press, 2006.

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.

Heeyoung Lee, Angel X. Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Deterministic Coreference Resolution Based on Entity-Centric, Precision-Ranked Rules. *Computational Linguistics*, 39(4):885–916, 2013.

Jay Yoon Lee, Sanket Vaibhav Mehta, Michael D. Wick, Jean-Baptiste Tristan, and Jaime G. Carbonell. Gradient-based Inference for Networks with Output Constraints. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Qi Li and Heng Ji. Incremental Joint Extraction of Entity Mentions and Relations. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, 2014.

Percy Liang, Hal Daumé, III, and Dan Klein. Structure Compilation: Trading Structure for Features. In *Proceedings of the 25th International Conference on Machine Learning*, pages 592–599, 2008.

Tie-Yan Liu. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

Xiaoqiang Luo. On Coreference Resolution Performance Metrics. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

Chao Ma, Janardhan Rao Doppa, John Walker Orr, Prashanth Mannem, Xiaoli Z. Fern, Thomas G. Dietterich, and Prasad Tadepalli. Prune-and-Score: Learning for Greedy Coreference Resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Chao Ma, Janardhan Rao Doppa, Prasad Tadepalli, Hamed Shahbazi, and Xiaoli Fern. Multi-Task Structured Prediction for Entity Analysis: Search-Based Learning Algorithms. In *Proceedings of The Asian Conference on Machine Learning (ACML)*, pages 514–529, 2017.

Chao Ma, F A Rezaur Rahman Chowdhury, Aryan Deshwal, Md Rakibul Islam, Janardhan Rao Doppa, and Dan Roth. Randomized Greedy Search for Structured Prediction: Amortized Inference and Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Andrew Mccallum and Ben Wellner. Toward Conditional Models of Identity Uncertainty with Application to Proper Noun Coreference. In *Proceedings of Advances in Neural Information Processing Systems*, pages 905–912. MIT Press, 2003.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2013.

MUC6. Coreference Task Definition. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 335–344, 1995.

Vincent Ng and Claire Cardie. Improving Machine Learning Approaches to Coreference Resolution. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 104–111, 2002.

NIST. The ACE Evaluation Plan, 2004.

NIST. The ACE Evaluation Plan, 2005.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Modeling Multilingual Unrestricted Coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, 2012.

Altaf Rahman and Vincent Ng. Coreference Resolution with World Knowledge. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 814–824, 2011.

Altaf Rahman and Vincent Ng. Narrowing the Modeling Gap: A Cluster-Ranking Approach to Coreference Resolution. *Journal of Artificial Intelligence Research (JAIR)*, 40:469–521, 2011.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence Level Training with Recurrent Neural Networks. *CoRR*, abs/1511.06732, 2016.

Lev Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2009.

Lev-Arie Ratinov and Dan Roth. Learning-based Multi-Sieve Co-reference Resolution with Knowledge. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP) Conference*, pages 1234–1244, 2012.

Stéphane Ross and Drew Bagnell. Efficient Reductions for Imitation Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010.

Stéphane Ross, Geoffery Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635, 2011.

Dan Roth and Wen-tau Yih. Integer Linear Programming Inference for Conditional Random Fields. In *Proceedings of International Conference on Machine Learning (ICML)*, 2005.

Rajhans Samdani and Dan Roth. Efficient Decomposed Learning for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2012.

Sameer Singh, Sebastian Riedel, Brian Martin, Jiaping Zheng, and Andrew McCallum. Joint Inference of Entities, Relations, and Coreference. In *AKBC Workshop*, 2013.

Wee Meng Soon, Daniel Chung, Daniel Chung Yong Lim, Yong Lim, and Hwee Tou Ng. A Machine Learning Approach to Coreference Resolution of Noun Phrases, 2001.

Russell Stewart and Stefano Ermon. Label-free Supervision of Neural Networks with Physics and Domain Knowledge. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 2576–2582, 2017.

Veselin Stoyanov and Jason Eisner. Easy-first Coreference Resolution. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pages 2519–2534, 2012.

Veselin Stoyanov, Claire Cardie, Nathan Gilbert, Ellen Riloff, David Buttler, and David Hysom. Coreference Resolution with Reconcile. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, 2010.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Bowen Tan, Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric P. Xing. Connecting the Dots Between MLE and RL for Sequence Generation, 2019.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *Proceedings of International Conference on Machine Learning (ICML)*, 2004.

Lifu Tu and Kevin Gimpel. Learning Approximate Inference Networks for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Marc B. Vilain, John D. Burger, John S. Aberdeen, Dennis Connolly, and Lynette Hirschman. A Model-theoretic Coreference Scoring Scheme. In *Proceedings of the 6th Conference on Message Understanding*, 1995.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph and Text Jointly Embedding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

David Weiss and Ben Taskar. Structured Prediction Cascades. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010.

Michael L. Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. SampleRank: Training Factor Graphs with Atomic Gradients. In *Proceedings of International Conference on Machine Learning (ICML)*, 2011.

Michael L. Wick, Sameer Singh, and Andrew McCallum. A Discriminative Hierarchical Model for Fast Coreference at Large Scale. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, pages 379–388, 2012.

Sam Wiseman and Alexander M. Rush. Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1296–1306, 2016.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning Linear Ranking Functions for Beam Search with Application to Planning. *Journal of Machine Learning Research (JMLR)*, 10:1571–1610, 2009.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 5502–5511. PMLR, 2018.

Chun-Nam John Yu and Thorsten Joachims. Learning Structural SVMs with Latent Variables. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

Jiaping Zheng, Luke Vilnis, Sameer Singh, Jinho D. Choi, and Andrew McCallum. Dynamic Knowledge-Base Alignment for Coreference Resolution. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2013.