Journal of Artificial Intelligence Research 50 (2014) 369-407 Submitted 10/13; published 6/14

$\mathcal{HC}$-Search: A Learning Framework for Search-based Structured Prediction

Janardhan Rao Doppa DOPPA@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University
Corvallis, OR 97331-5501, USA

Alan Fern AFERN@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University
Corvallis, OR 97331-5501, USA

Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of EECS, Oregon State University
Corvallis, OR 97331-5501, USA

Abstract

Structured prediction is the problem of learning a function that maps structured inputs to structured outputs. Prototypical examples of structured prediction include part-of-speech tagging and semantic segmentation of images. Inspired by the recent successes of search-based structured prediction, we introduce a new framework for structured prediction called $\mathcal{HC}$-Search. Given a structured input, the framework uses a search procedure guided by a learned heuristic $\mathcal{H}$ to uncover high quality candidate outputs and then employs a separate learned cost function $\mathcal{C}$ to select a final prediction among those outputs. The overall loss of this prediction architecture decomposes into the loss due to $\mathcal{H}$ not leading to high quality outputs, and the loss due to $\mathcal{C}$ not selecting the best among the generated outputs. Guided by this decomposition, we minimize the overall loss in a greedy stage-wise manner by first training $\mathcal{H}$ to quickly uncover high quality outputs via imitation learning, and then training $\mathcal{C}$ to correctly rank the outputs generated via $\mathcal{H}$ according to their true losses. Importantly, this training procedure is sensitive to the particular loss function of interest and the time-bound allowed for predictions. Experiments on several benchmark domains show that our approach significantly outperforms several state-of-the-art methods.

1. Introduction

We consider the problem of structured prediction, where the predictor must produce a structured output given a structured input. For example, in Part-Of-Speech (POS) tagging, the structured input is a sequence of words and structured output corresponds to the POS tags for those words. Image scene labeling is another example, where the structured input is an image and the structured output is a semantic labeling of the image regions. Structured prediction tasks such as above arise in several domains ranging from natural language processing (e.g., named entity recognition, coreference resolution, and semantic parsing) and computer vision (e.g., multi-object tracking and activity recognition in videos) to speech (e.g., text-to-speech mapping and speech recognition) and compuational biology (e.g., protein secondary structure prediction and gene prediction).

Viewed as a traditional classification problem, the set of possible classes in structured prediction is exponential in the size of the input. Thus, the problem of producing an

output is combinatorial in nature, which introduces the non-trivial choice of selecting a computational framework for producing outputs. Importantly, this framework needs to balance two conflicting criteria: 1) It must be flexible enough to allow for complex and accurate structured predictors to be learned, and 2) It must support inference of outputs within the computational time constraints of an application. One of the core research challenges in structured prediction has been to achieve a balance between these criteria.

A standard approach to structured prediction is to learn a cost function $\mathcal{C}(\mathbf{x}, \mathbf{y})$ for scoring a potential structured output $\mathbf{y}$ given a structured input $\mathbf{x}$. Given such a cost function and a new input $\mathbf{x}$, the output computation involves solving the so-called "Argmin" problem, which is to find the minimum cost output for a given input.

$$\hat{\mathbf{y}} = \arg\min_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \ \mathcal{C}(x, y) \tag{1}$$

For example, approaches such as Conditional Random Fields (CRFs) (Lafferty, McCallum, & Pereira, 2001), Max-Margin Markov Networks (Taskar, Guestrin, & Koller, 2003) and Structured SVMs (Tsochantaridis, Hofmann, Joachims, & Altun, 2004) represent the cost function as a linear model over template features of both $\mathbf{x}$ and $\mathbf{y}$. Unfortunately, exactly solving the Argmin problem is often intractable. Efficient solutions exist only in limited cases such as when the dependency structure among features forms a tree. In such cases, one is forced to simplify the features to allow for tractable inference, which can be detrimental to prediction accuracy. Alternatively, a heuristic optimization method can be used such as loopy belief propagation or variational inference. While such methods have shown some success in practice, it can be difficult to characterize their solutions and to predict when they are likely to work well for a new problem.

We are inspired by the recent successes of output-space search approaches (Doppa, Fern, & Tadepalli, 2012; Wick, Rohanimanesh, Bellare, Culotta, & McCallum, 2011), which place few restrictions on the form of the cost function. These methods learn and use a cost function to conduct a search through the space of complete outputs via a search procedure (e.g., greedy search), and return the least cost output that is uncovered during the search as the prediction. The search procedure only needs to be able to efficiently evaluate the cost function at specific input-output pairs, which is generally straightforward even when the corresponding Argmin problem is intractable. Thus, these methods are free to increase the complexity of the cost function without considering its impact on the inference complexity.

While these approaches have achieved state-of-the-art performance on a number of benchmark problems, a primary contribution of this paper is to highlight a fundamental deficiency that they share. In particular, prior work uses a single cost function to serve the dual roles of both: 1) guiding the search toward good outputs, and 2) scoring the generated outputs in order to select the best one. Serving these dual roles often means that the cost function needs to make unclear tradeoffs, increasing the difficulty of learning. Indeed, in the traditional AI search literature, these roles are typically served by different functions, mainly a heuristic function for guiding search, and a cost/evaluation function (often part of the problem definition) for selecting the final output.

In this paper, we study a new framework for structured prediction called $\mathcal{HC}$-Search that closely follows the traditional search literature. The key idea is to learn distinct functions for each of the above roles: 1) a *heuristic function* $\mathcal{H}$ to guide the search and generate a set of high-quality candidate outputs, and 2) a *cost function* $\mathcal{C}$ to score the outputs generated

by the heuristic $\mathcal{H}$. Given a structured input, predictions are made by using $\mathcal{H}$ to guide a search strategy (e.g., greedy search or beam search) until a time bound to generate a set of candidate outputs and then returning the generated output of least cost according to $\mathcal{C}$.

While the move to $\mathcal{HC}$-Search might appear to be relatively small, there are significant implications in terms of both theory and practice. First, the regret of the $\mathcal{HC}$-Search approach can be decomposed into the loss due to $\mathcal{H}$ not leading to high quality outputs, and the loss due to $\mathcal{C}$ not selecting the best among the generated outputs. This decomposition helps us target our training to minimize each of these losses individually in a greedy stage-wise manner. Second, as we will show, the performance of the approaches with a single function can be arbitrarily bad when compared to that of $\mathcal{HC}$-Search in the worst case. Finally, we show that in practice $\mathcal{HC}$-Search performs significantly better than the single cost function search and other state-of-the-art approaches to structured prediction.

The effectiveness of the $\mathcal{HC}$-Search approach for a particular problem depends critically on: 1) the quality of the search space over complete outputs being used, where quality is defined as the expected depth at which target outputs (zero loss outputs) can be located, 2) our ability to learn a heuristic function for effectively guiding the search to generate high-quality candidate outputs, and 3) the accuracy of the learned cost function in selecting the best output among the candidate outputs generated by the heuristic function. In this work, we assume the availability of an efficient search space over complete outputs and provide an effective training regime for learning both heuristic function and cost function within the $\mathcal{HC}$-Search framework.

1.1 Summary of Contributions

The main contributions of our work are as follows: 1) We introduce the $\mathcal{HC}$-Search framework, where two different functions are learned to serve the purposes of search heuristic and cost function as in the search literature; 2) We analyze the representational power and computational complexity of learning within the $\mathcal{HC}$-Search framework; 3) We identify a novel decomposition of the overall regret of the $\mathcal{HC}$-Search approach in terms of *generation loss*, the loss due to heuristic not generating high-quality candidate outputs, and *selection loss*, the loss due to cost function not selecting the best among the generated outputs; 4) Guided by the decomposition, we propose a stage-wise approach to learning the heuristic and cost functions based on imitation learning; 5) We empirically evaluate the $\mathcal{HC}$-Search approach on a number of benchmarks, comparing it to state-of-the-art methods and analyzing different dimensions of the framework.

The remainder of the paper proceeds as follows. In Section 2, we introduce our problem setup, give a high-level overview of our framework, and analyze the complexity of $\mathcal{HC}$-Search learning problem. We describe our approaches to heuristic and cost function learning in Section 3. Section 4 presents our experimental results followed by an engineering methodology for applying our framework to new problems in Section 5. Finally, Sections 6 and 7 discuss related work and future directions.

2. $\mathcal{HC}$-Search Framework

In this section, we first state the formal problem setup and then describe the specifics of the search spaces and search strategies that we will investigate in this work. Next, we give a high-level overview of our $\mathcal{HC}$-Search framework along with its learning objective.

2.1 Problem Setup

A structured prediction problem specifies a space of structured inputs $\mathcal{X}$, a space of structured outputs $\mathcal{Y}$, and a non-negative *loss function* $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re^+$ such that $L(x, y', y^*)$ is the loss associated with labeling a particular input x by output y' when the true output is y^*. We are provided with a training set of input-output pairs $\{(x, y^*)\}$ drawn from an unknown target distribution $\mathcal{D}$. The goal is to return a function/predictor from structured inputs to outputs whose predicted outputs have low expected loss with respect to the distribution $\mathcal{D}$. Since our algorithms will be learning heuristic and cost functions over input-output pairs, as is standard in structured prediction, we assume the availability of a *feature function* $\Phi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^n$ that computes an n dimensional feature vector for any pair. Importantly, we can employ two different feature functions $\Phi_{\mathcal{H}}$ and $\Phi_{\mathcal{C}}$ for heuristic and cost function noting that they are serving two different roles: the heuristic is making local decisions to guide the search towards high-quality outputs and the cost function is making global decisions by scoring the candidate outputs generated by the heuristic in this framework.

2.2 Search Spaces and Search Strategies

We overview some basic search concepts in the context of our search-based framework below.

2.2.1 Search Spaces

Our approach is based on search in a space $\mathcal{S}_o$ of complete outputs, which we assume to be given. Every state in a search space over complete outputs consists of an input-output pair (x, y), representing the possibility of predicting y as the output for structured input x. Such a search space is defined in terms of two functions: 1) An *initial state function* I such that $I(x)$ returns an initial state for input x, and 2) a *successor function* S such that for any search state (x, y), $S((x, y))$ returns a set of next states $\{(x, y_1), \cdots, (x, y_k)\}$ that share the same input x as the parent. For example, in a sequence labeling problem, such as part-of-speech tagging, (x, y) is a sequence of words and corresponding part-of-speech (POS) labels. The successors of (x, y) might correspond to all ways of changing one of the output labels in y, the so-called "flipbit" space. Figure 1 provides an illustration of the flipbit search space for the handwriting recognition task.

Search Space Quality. The effectiveness of our $\mathcal{HC}$-Search framework depends on the quality of the search space that is used. The quality of a search space can in turn be understood in terms of the expected amount of search needed to uncover the correct output y^*. For most search procedures, the time required to find a target output y^* will grow as a function of the depth of the target. Thus, one way to quantify the expected amount of search, independently of the specific search strategy, is by considering the expected depth of target outputs y^*. In particular, for a given input-output pair (x, y^*), the target depth d

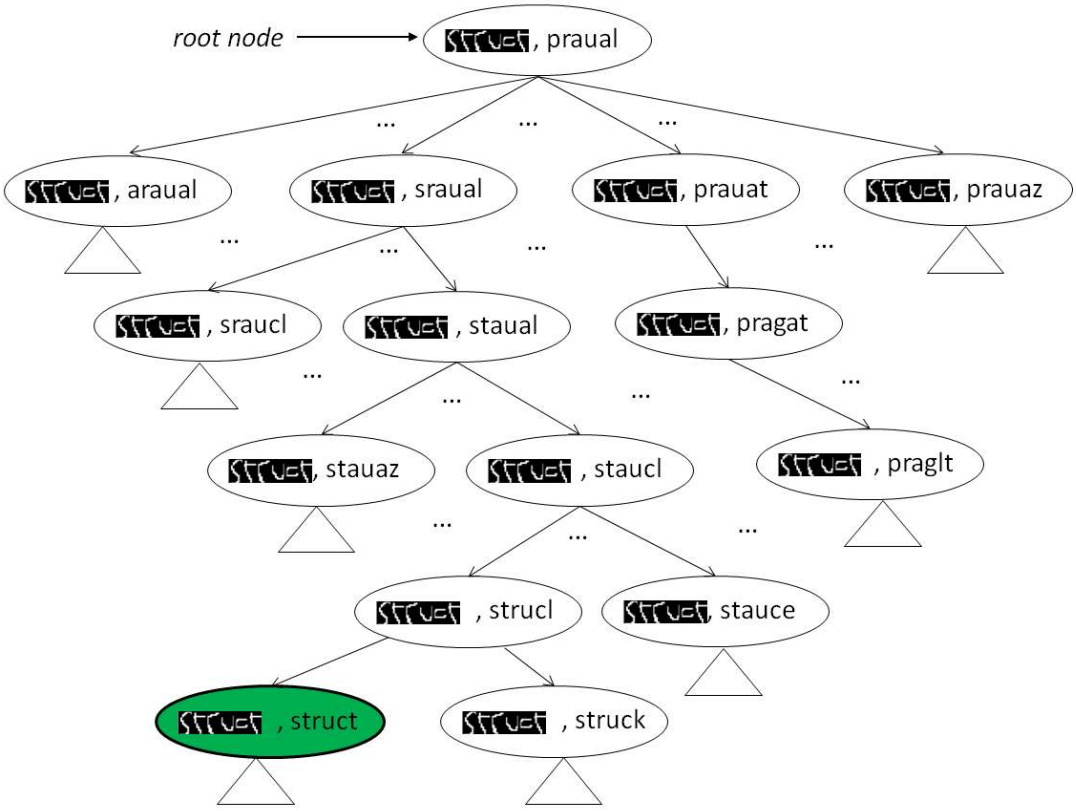

Figure 1: An example Flipbit search space for the handwriting recognition problem. Each search state consists of a complete input-output pair and the complete output at every state differs from that of its parent by exactly one label. The highlighted state corresponds to the one with true output y^* at the smallest depth, which is equal to the number of errors in the initial state.

is defined as the minimum depth at which we can find a state corresponding to the target output y^* ($d = 5$ in the example flipbit space shown in Figure 1). Clearly according to this definition, the expected target depth of the flipbit space is equal to the expected number of errors in the output corresponding to the initial state.

A variety of search spaces, such as the above flipbit space, Limited Discrepancy Search (LDS) space (Doppa et al., 2012), and those defined based on hand-designed proposal distributions (Wick et al., 2011) have been used in past research. While our work applies to any such space, we will focus on the LDS space in our experiment, which has been shown to effectively uncover high-quality outputs at relatively shallow search depths (Doppa et al., 2012).

The LDS space is defined in terms of a recurrent classifier h which uses the next input token, e.g. word, and output tokens in a small preceding window, e.g. POS labels, to predict the next output token. The initial state of the LDS space consists of the input x paired with the output of the recurrent classifier h on x. One problem with recurrent classifiers is that when a recurrent classifier makes a mistake, its effects get propagated to down-stream tokens. The LDS space is designed to prevent this error propagation by immediately correcting the mistakes made before continuing with the recurrent classifier. Since we do not know where the mistakes are made and how to correct them, all possible corrections, called *discrepancies*, are considered. Hence the successors of any state (x, y) in the LDS space consist of the results of running the recurrent classifier after changing exactly one more label, i.e., introducing a single new discrepancy, somewhere in the current output sequence y while preserving all previously introduced discrepancies. In previous work, the LDS space has been shown to be effective in uncovering high-quality outputs at relatively shallow search depths, as one would expect with a good recurrent classifier (Doppa et al., 2012). The Appendix contains more details and examples of the LDS space we employ in this work.

2.2.2 Search Strategies

Recall that in our $\mathcal{HC}$-Search framework, the role of the search procedure is to uncover high-quality outputs. We can consider both uninformed and informed search strategies. However, uninformed search procedures like depth bounded breadth-first search will only be practical when high-quality outputs exist at small depths and even when they are feasible, they are not a good choice because they don't use the search time bound in an intelligent way to make predictions. For most structured prediction problems, informed search strategies that take heuristic functions into account, such as greedy search or best-first search are a better choice, noting that their effectiveness depends the quality of the search heuristic $\mathcal{H}$. Prior work (Doppa et al., 2012; Wick et al., 2011) has shown that greedy search (hill climbing based on the heuristic value) works quite well for a number of structured prediction tasks when used with an effective search space. Thus, in this work, we focus our empirical work on the $\mathcal{HC}$-Search framework using greedy search, though the approach applies more widely.

2.3 $\mathcal{HC}$-Search Approach

Our approach is parameterized by a search space over complete outputs $\mathcal{S}_\wr$ (e.g., LDS space), a heuristic search strategy $\mathcal{A}$ (e.g., greedy search), a learned heuristic function

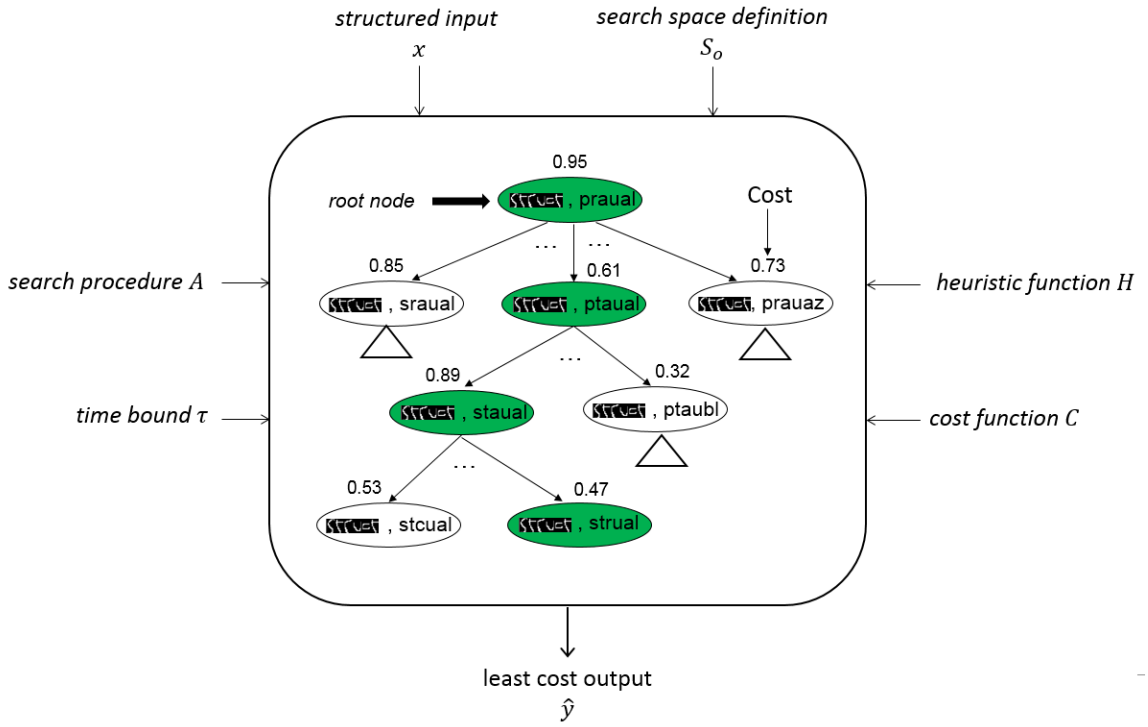

Figure 2: A high level overview of our $\mathcal{HC}$-Search framework. Given a structured input x and a search space definition S_o, we first instantiate a search space over complete outputs. Each search node in this space consists of a complete input-output pair. Next, we run a search procedure $\mathcal{A}$ (e.g., greedy search) guided by the heuristic function $\mathcal{H}$ for a time bound τ. The highlighted nodes correspond to the search trajectory traversed by the search procedure, in this case greedy search. The scores on the nodes correspond to cost values, which are different from heuristic scores (not shown in the figure). We return the least cost output $\hat{y}$ that is uncovered during the search as the prediction for input x.

$\mathcal{H} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$, and a learned cost function $\mathcal{C} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$. Given an input x and a prediction time bound τ, $\mathcal{HC}$-Search makes predictions as follows. It traverses the search space starting at $I(x)$ using the search procedure $\mathcal{A}$ guided by the heuristic function $\mathcal{H}$ until the time bound is exceeded. Then the cost function $\mathcal{C}$ is applied to find return the least-cost output $\hat{y}$ that is generated during the search as the prediction for input x. Figure 2 gives a high-level overview of our $\mathcal{HC}$-Search framework.

More formally, let $\mathcal{Y}_{\mathcal{H}}(x)$ be the set of candidate outputs generated using heuristic $\mathcal{H}$ for a given input x. The output returned by $\mathcal{HC}$-Search is $\hat{y}$ the least cost output in this set according to $\mathcal{C}$, i.e.,

$$\hat{y} = \arg\min_{y \in \mathcal{Y}_{\mathcal{H}}(x)} \; \mathcal{C}(x, y)$$

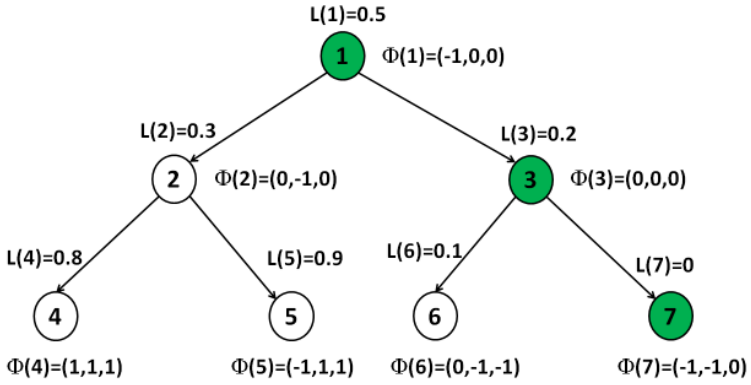

Figure 3: An example that illustrates that $\mathcal{C}$-Search can suffer arbitrarily large loss compared to $\mathcal{HC}$-Search.

The expected loss of the $\mathcal{HC}$-Search approach $\mathcal{E}(\mathcal{H},\mathcal{C})$ for a given heuristic $\mathcal{H}$ and $\mathcal{C}$ can be defined as

$$\mathcal{E}\left(\mathcal{H},\mathcal{C}\right) = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ L\left(x,\hat{y},y^*\right) \tag{2}$$

Our goal is to learn a heuristic function $\mathcal{H}^*$ and corresponding cost function $\mathcal{C}^*$ that minimize the expected loss from their respective spaces $\mathbf{H}$ and $\mathbf{C}$, i.e.,

$$(\mathcal{H}^*,\mathcal{C}^*) = \arg\min_{(\mathcal{H},\mathcal{C})\in\mathbf{H}\times\mathbf{C}}\ \ \mathcal{E}\left(\mathcal{H},\mathcal{C}\right) \tag{3}$$

In contrast to our framework, existing approaches for output space search (Doppa et al., 2012; Wick et al., 2011) use a single function (say $\mathcal{C}$) to serve the dual purpose of heuristic and cost function. This raises the question of whether $\mathcal{HC}$-Search, which uses two different functions, is strictly more powerful in terms of its achievable losses. The following proposition shows that the expected loss of $\mathcal{HC}$-Search can be arbitrarily smaller than when restricting to using a single function $\mathcal{C}$.

Proposition 1. *Let $\mathcal{H}$ and $\mathcal{C}$ be functions from the same function space. Then for all learning problems, $\min_{\mathcal{C}} \mathcal{E}(\mathcal{C},\mathcal{C}) \geq \min_{(\mathcal{H},\mathcal{C})} \mathcal{E}(\mathcal{H},\mathcal{C})$. Moreover there exist learning problems for which $\min_{\mathcal{C}} \mathcal{E}(\mathcal{C},\mathcal{C})$ is arbitrarily larger (i.e. worse) than $\min_{(\mathcal{H},\mathcal{C})} \mathcal{E}(\mathcal{H},\mathcal{C})$.*

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.

To see the second part, consider a problem with a single training instance with search space shown in Figure 3. The search procedure will be greedy search that is either guided by $\mathcal{H}$ for $\mathcal{HC}$-Search, or by $\mathcal{C}$ when only one function is used. $L(n)$ and $\Phi(n)$ represents the true loss and the feature vector of node n respectively. The cost and heuristic functions are linear functions of $\Phi(n)$. Node 7 corresponds to the lowest-loss output and greedy search must follow the trajectory of highlighted nodes in order to reach that output. First consider $\mathcal{HC}$-Search. For the highlighted path to be followed the heuristic $\mathcal{H}$ needs to satisfy the following constraints: $\mathcal{H}(3)<\mathcal{H}(2)$, $\mathcal{H}(7)<\mathcal{H}(6)$, and the weights $w_{\mathcal{H}} = [-1,1,1]$ result in a

heuristic that satisfies the constraints. Given this heuristic function, in order to return node 7 as the final output, the cost function must satisfy the following constraints: $\mathcal{C}(7)<\mathcal{C}(1)$, $\mathcal{C}(7)<\mathcal{C}(2)$, $\mathcal{C}(7)<\mathcal{C}(3)$, $\mathcal{C}(7)<\mathcal{C}(6)$, and the weights $w_{\mathcal{C}} = [-1, -1, 0]$ solve the problem. Thus we see that $\mathcal{HC}$-Search can achieve zero loss on this problem.

Now consider the case where a single function $\mathcal{C}$ is used for the heuristic and cost function. Here in order to generate a loss of zero, the function $\mathcal{C}$ must satisfy the combined set of constraints from above that were placed on the heuristic and cost function. However, it can be verified that there is no set of weights that satisfies both $\mathcal{C}(3)<\mathcal{C}(2)$ and $\mathcal{C}(7)<\mathcal{C}(1)$, and hence, there is no single function $\mathcal{C}$ in our space that can achieve a loss of zero. By scaling the losses by constant factors we can make the loss suffered arbitrarily high.

□

Thus, we see that there can be potential representational advantages to following the $\mathcal{HC}$-Search framework. In what follows, we consider the implications of this added expressiveness in terms of the worst-case time complexity of learning.

2.4 Learning Complexity

We now consider the feasibility of efficient, optimal learning in the simplest setting of greedy search using linear heuristic and cost functions represented by their weight vectors $w_{\mathcal{H}}$ and $w_{\mathcal{C}}$ respectively. In particular, we consider the *$\mathcal{HC}$-Search Consistency Problem*, where the input is a training set of structured examples, and we must decide whether or not there exists $w_{\mathcal{H}}$ and $w_{\mathcal{C}}$ such that $\mathcal{HC}$-Search using greedy search will achieve zero loss on the training set. We first note, that this problem can be shown to be NP-Hard by appealing to results on learning for beam search (Xu, Fern, & Yoon, 2009a). In particular, results there imply that in all but trivial cases, simply determining whether or not there is a linear heuristic $w_{\mathcal{H}}$ that uncovers a zero loss search node is NP-Hard. Since $\mathcal{HC}$-Search can only return zero loss outputs when the heuristic is able to uncover them, we see that our problem is also hard.

Here we prove a stronger result that provides more insight into the $\mathcal{HC}$-Search framework. In particular, we show that even when it is "easy" to learn a heuristic that uncovers all zero loss outputs, the consistency problem is still hard. This shows, that in the worst case the hardness of our learning problem is not simply a result of the hardness of discovering good outputs. Rather our problem is additionally complicated by the potential interaction between $\mathcal{H}$ and $\mathcal{C}$. Intuitively, when learning $\mathcal{H}$ in the worst case there can be ambiguity about which of many small loss outputs to generate, and for only some of those will we be able to find an effective $\mathcal{C}$ to return the best one. This is formalized by the following theorem, whose proof is in the Appendix.

Theorem 1. *The $\mathcal{HC}$-Search Consistency Problem for greedy search and linear heuristic and cost functions is NP-Hard even when we restrict to problems for which all possible heuristic functions uncover a zero loss output.*

3. Learning Approach

The above complexity result suggests that, in general, learning the optimal $(\mathcal{H}^*, \mathcal{C}^*)$ pair is impractical due to their potential interdependence. In this section, we develop a greedy

stage-wise learning approach that first learns $\mathcal{H}$ and then a corresponding $\mathcal{C}$. The approach is motivated by observing a decomposition of the expected loss into components due to $\mathcal{H}$ and $\mathcal{C}$. Below, we first describe the decomposition and the staged learning approach that it motivates. Next we describe our approaches for learning the heuristic and cost functions.

3.1 Loss Decomposition and Staged Learning

For any heuristic $\mathcal{H}$ and cost function $\mathcal{C}$, the expected loss $\mathcal{E}(\mathcal{H}, \mathcal{C})$ can be decomposed into two parts: 1) the *generation loss* $\epsilon_{\mathcal{H}}$, due to $\mathcal{H}$ not generating high-quality outputs, and 2) the *selection loss* $\epsilon_{\mathcal{C}|\mathcal{H}}$, the additional loss (conditional on $\mathcal{H}$) due to $\mathcal{C}$ not selecting the best loss output generated by the heuristic. Formally, let $y^*_{\mathcal{H}}$ be the best loss output in the set $\mathcal{Y}_{\mathcal{H}}(x)$, i.e.,

$$y^*_{\mathcal{H}} = \arg\min_{y \in \mathcal{Y}_{\mathcal{H}}(x)} \; L(x, y, y^*)$$

We can express the decomposition as follows:

$$\mathcal{E}(\mathcal{H}, \mathcal{C}) = \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L(x, y^*_{\mathcal{H}}, y^*)}_{\epsilon_{\mathcal{H}}} + \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L(x, \hat{y}, y^*) - L(x, y^*_{\mathcal{H}}, y^*)}_{\epsilon_{\mathcal{C}|\mathcal{H}}} \tag{4}$$

Note that given labeled data, it is straightforward to estimate both the generation and selection loss, which is useful for diagnosing the $\mathcal{HC}$-Search framework. For example, if one observes that a system has high generation loss, then there will be little payoff in working to improve the cost function. In our empirical evaluation we will further illustrate how the decomposition is useful for understanding the results of learning.

In addition to being useful for diagnosis, the decomposition motivates a learning approach that targets minimizing each of the errors separately. In particular, we optimize the overall error of the $\mathcal{HC}$-Search approach in a greedy stage-wise manner. We first train a heuristic $\hat{\mathcal{H}}$ in order to optimize the generation loss component $\epsilon_{\mathcal{H}}$ and then train a cost function $\hat{\mathcal{C}}$ to optimize the selection loss $\epsilon_{\mathcal{C}|\hat{\mathcal{H}}}$ conditioned on $\hat{\mathcal{H}}$.

$$\hat{\mathcal{H}} \approx \arg\min_{\mathcal{H} \in \mathbf{H}} \; \epsilon_{\mathcal{H}}$$
$$\hat{\mathcal{C}} \approx \arg\min_{\mathcal{C} \in \mathbf{C}} \; \epsilon_{\mathcal{C}|\hat{\mathcal{H}}}$$

Note that this approach is greedy in the sense that $\hat{\mathcal{H}}$ is learned without considering the implications for learning $\hat{\mathcal{C}}$. While the proof of Theorem 1 hinges on this coupling, we have found that in practice, learning $\hat{\mathcal{H}}$ independently of $\hat{\mathcal{C}}$ is a very effective strategy.

In what follows, we first describe a generic approach for heuristic function learning that is applicable for a wide range of search spaces and search strategies, and then explain our cost function learning algorithm.

3.2 Heuristic Function Learning

Most generally, learning a heuristic can be viewed as a Reinforcement Learning (RL) problem where the heuristic is viewed as a policy for guiding "search actions" and rewards

are received for uncovering high quality outputs (Zhang & Dietterich, 1995). In fact, this approach has been explored for structured prediction in the case of greedy search (Wick, Rohanimanesh, Singh, & McCallum, 2009) and was shown to be effective given a carefully designed reward function and action space. While this is a viable approach, general purpose RL can be quite sensitive to the algorithm parameters and specific definition of the reward function and actions, which can make designing an effective learner quite challenging. Indeed, recent work (Jiang, Teichert, Daumé III, & Eisner, 2012), has shown that generic RL algorithms can struggle for some structured prediction problems, even with significant effort put forth by the designer. Hence, in this work, we follow an approach based on imitation learning, that makes stronger assumptions, but has nevertheless been very effective and easy to apply across a variety of problems.

Algorithm 1 Heuristic Function Learning via Exact Imitation

Input: $\mathcal{D}$ = Training examples, (I, S) = Search space definition, L = Loss function, $\mathcal{A}$ = Rank-based search procedure, τ_{max} = search time bound
Output: $\mathcal{H}$, the heuristic function

Initialize the set of ranking examples $\mathcal{R} = \emptyset$
for each training example $(x, y^*) \in \mathcal{D}$ **do**
  $s_0 = I(x)$ *// initial state of the search tree*
  $M_0 = \{s_0\}$ *// set of open nodes in the internal memory of the search procedure*
  for each search step $t = 1$ to τ_{max} **do**
    Select the state(s) to expand: $N_t =$**Select**$(\mathcal{A}, L, M_{t-1})$
    Expand every state $s \in N_t$ using the successor function S: $C_t =$**Expand**(N_t, S)
    Prune states and update the internal memory state of the search procedure: $M_t =$**Prune**$(\mathcal{A}, L, M_{t-1} \cup C_t \setminus N_t)$
    Generate ranking examples R_t to imitate this search step
    Add ranking examples R_t to $\mathcal{R}$: $\mathcal{R} = \mathcal{R} \cup R_t$ *// aggregation of training data*
  end for
end for
$\mathcal{H} =$**Rank-Learner**$(\mathcal{R})$ *// learn heuristic function from all the ranking examples*
return learned heuristic function $\mathcal{H}$

Our heuristic learning approach is based on the observation that for many structured prediction problems, we can quickly generate very high-quality outputs by guiding the search procedure using the true loss function L as a heuristic. Obviously this can only be done for the training data for which we know y^*. This suggests formulating the heuristic learning problem in the framework of *imitation learning* by attempting to learn a heuristic that mimics the search decisions made by the true loss function on training examples. The learned heuristic need not approximate the true loss function uniformly over the output space, but need only make the distinctions that were important for guiding the search. The main assumptions made by this approach are: 1) the true loss function can provide effective heuristic guidance to the search procedure, so that it is worth imitating, and 2) we can learn to imitate those search decisions sufficiently well.

This imitation learning approach is similar to prior work on learning single cost functions for output-space search (Doppa et al., 2012). However, a key distinction here is that learning

is focused on only making distinctions necessary for uncovering good outputs (the purpose of the heuristic) and hence requires a different formulation. As in prior work, in order to avoid the need to approximate the loss function arbitrarily closely, we restrict ourselves to "rank-based" search strategies. A search strategy is called *rank-based* if it makes all its search decisions by comparing the *relative values* of the search nodes (their ranks) assigned by the heuristic, rather than being sensitive to absolute values of heuristic. Most common search procedures such as greedy search, beam search, and best-first search fall under this category.

3.2.1 Imitating Search Behavior

Given a search space over complete outputs S, a rank-based search procedure $\mathcal{A}$, and a search time bound τ, our learning procedure generates imitation training data for each training example (x, y^*) as follows. We run the search procedure $\mathcal{A}$ for a time bound of τ for input x using a heuristic equal to the true loss function, i.e. $\mathcal{H}(x, y) = L(x, y, y^*)$. During the search process we observe all of the pairwise ranking decisions made by $\mathcal{A}$ using this oracle heuristic and record those that are sufficient (see below) for replicating the search. If the state (x, y_1) has smaller loss than (x, y_2), then a ranking example is generated in the form of the constraint $\mathcal{H}(x, y_1) < \mathcal{H}(x, y_2)$. Ties are broken using a fixed arbitrator[1]. The aggregate set of ranking examples collected over all the training examples is then given to a learning algorithm to learn the weights of the heuristic function.

If we can learn a function $\mathcal{H}$ from hypothesis space $\mathbf{H}$ that is consistent with these ranking examples, then the learned heuristic is guaranteed to replicate the oracle-guided search on the training data. Further, given assumptions on the base learning algorithm (e.g. PAC), generic imitation learning results can be used to give generalization guarantees on the performance of search on new examples (Khardon, 1999; Fern, Yoon, & Givan, 2006; Syed & Schapire, 2010; Ross & Bagnell, 2010). Our experiments show, that the simple approach described above, performs extremely well on our problems.

Algorithm 1 describes our approach for heuristic function learning via exact imitation of search guided by the loss function. It is applicable to a wide-range of search spaces, search procedures and loss functions. The learning algorithm takes as input: 1) $\mathcal{D} = \{(x, y^*)\}$, a set of training examples for a structured prediction problem (e.g., handwriting recognition); 2) $\mathcal{S}_o = (I, S)$, a search space over complete outputs (e.g., LDS space), where I is the initial state function and S is the successor function; 3) L, a task loss function defined over complete outputs (e.g., hamming loss); 4) $\mathcal{A}$, a rank-based search procedure (e.g., greedy search); 5) τ_{max}, the search time bound (e.g., number of search steps).

The algorithmic description of Algorithm 1 assumes that the search procedure $\mathcal{A}$ can be described in terms of three steps that are executed repeatedly on an open list of search nodes: 1) selection, 2) expansion and 3) pruning. In each execution, the search procedure selects one or more open nodes from its internal memory for expansion (step 6) based on heuristic value, and expands all the selected nodes to generate the candidate set (step 7). It retains only a subset of all the open nodes after expansion in its internal memory and prunes away all the remaining ones (step 8) again based on heuristic value. For example,

1. For the LDS Space that we employed in this work, we implemented an arbitrator which breaks the ties based on the position of the discrepancy (prefers earlier discrepancies).

greedy search maintains only the best node, best-first beam search retains only the best b nodes for a fixed beam-width b, and pure best first search does not do any pruning.

Algorithm 1 loops through each training example and collects a set of ranking constraints. Specifically, for example (x, y^*), the search procedure is run for a time bound of τ_{max} using the true loss function L as the heuristic (steps 2-12). During each search step a set of pairwise ranking examples is generated that are sufficient for allowing the search step to be imitated (step 9) as described in more detail below. After all such constraints are aggregated across all search steps of all training examples, they are given to a rank-learning algorithm (e.g., Perceptron or SVM-Rank) to learn the weights of the heuristic function (step 13).

The most important step in our heuristic function learning algorithm is the generation of ranking examples to imitate each step of the search procedure (step 9). In what follows, we will give a generic description of "sufficient" pairwise decisions to imitate the search, and illustrate them for greedy search through a simple example.

3.2.2 Sufficient Pairwise Decisions

Above we noted that we only need to collect and learn to imitate the "sufficient" pairwise decisions encountered during search. We say that a set of constraints is sufficient for a structured training example (x, y^*), if any heuristic function that is consistent with the constraints causes the search to follow the same trajectory of open lists encountered during search. The precise specification of these constraints depends on the actual search procedure that is being used. For rank-based search procedures, the sufficient constraints can be categorized into two types:

1. *Selection* constraints, which ensure that the search node(s) from the internal memory state that will be expanded in the next search step is (are) ranked better than all other nodes.

2. *Pruning* constraints, which ensure that the internal memory state (set of search nodes) of the search procedure is preserved at every search step. More specifically, these constraints involve ranking every search node in the internal memory state better (lower $\mathcal{H}$-value) than those that are pruned.

Below, we will illustrate these constraints concretely for greedy search noting that similar formulations for other rank-based search procedures are straightforward (See (Doppa, Fern, & Tadepalli, 2014a) for beam search formulation).

3.2.3 Constraints for Greedy Search

This is the most basic rank-based search procedure. For a given input x, it traverses the search space by selecting the next state as the successor of the current state that looks best according to the heuristic function $\mathcal{H}$. In particular, if s_i is the search state at step i, greedy search selects $s_{i+1} = \arg\min_{s \in S(s_i)} \mathcal{H}(s)$, where $s_0 = I(x)$. In greedy search, the internal memory state of the search procedure at step i consists of only the best open (unexpanded) node s_i.

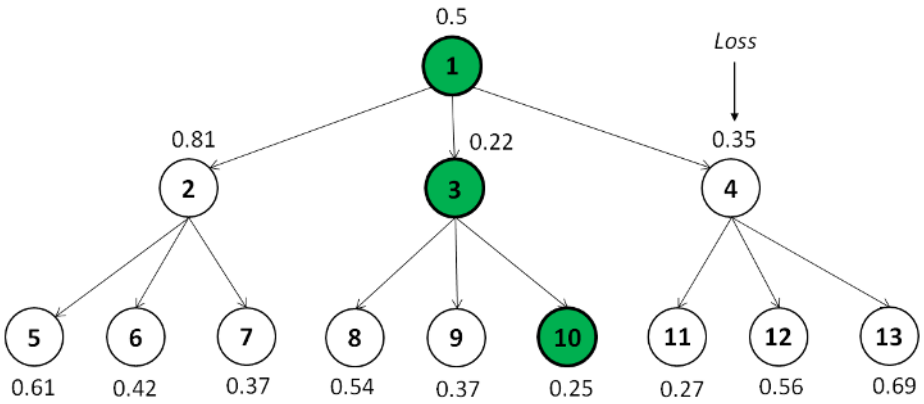

Figure 4: An example search tree that illustrates greedy search with loss function. Each node represents a complete input-output pair and can be evaluated using the loss function. The highlighted nodes correspond to the trajectory of greedy search guided by the loss function.

Let (x, y_i) correspond to the input-output pair associated with state s_i. Since greedy search maintains only a single open node s_i in its internal memory at every search step i, there are no selection constraints. Let C_{i+1} be the candidate set after expanding state s_i, i.e., $C_{i+1} = S(s_i)$. Let s_{i+1} be the best node in the candidate set C_{i+1} as evaluated by the loss function, i.e., $s_{i+1} = \arg\min_{s \in C_{i+1}} L(s)$. As greedy search prunes all the nodes in the candidate set other than s_{i+1}, pruning constraints need to ensure that s_{i+1} is ranked better than all the other nodes in C_{i+1}. Therefore, we include one ranking constraint for every node $(x, y) \in C_{i+1} \setminus (x, y_{i+1})$ such that $\mathcal{H}(x, y_{i+1}) < \mathcal{H}(x, y)$.

We will now illustrate these ranking constraints through an example. Figure 4 shows an example search tree of depth two with associated losses for every search node. The highlighted nodes correspond to the trajectory of greedy search with loss function that our learner has to imitate. At the first search step, $\{\mathcal{H}(3) < \mathcal{H}(2), \mathcal{H}(3) < \mathcal{H}(4)\}$ are the pruning constraints. Similarly, $\{\mathcal{H}(10) < \mathcal{H}(8), \mathcal{H}(10) < \mathcal{H}(9)\}$ form the pruning constraints at the second search step. Therefore, the aggregate set of constraints needed to imitate the greedy search behavior shown in Figure 4 are:
$\{\mathcal{H}(3) < \mathcal{H}(2), \mathcal{H}(3) < \mathcal{H}(4), \mathcal{H}(10) < \mathcal{H}(8), \mathcal{H}(10) < \mathcal{H}(9)\}$.

3.3 Cost Function Learning

Given a learned heuristic $\mathcal{H}$, we now want to learn a cost function that correctly ranks the potential outputs generated by the search procedure guided by $\mathcal{H}$. More formally, let $\mathcal{Y}_{\mathcal{H}}(x)$ be the set of candidate outputs generated by the search procedure guided by heuristic $\mathcal{H}$ for a given input x, and l_{best} be the loss of the best output among those outputs as evaluated by the true loss function L, i.e., $l_{best} = min_{y \in \mathcal{Y}_{\mathcal{H}}(x)} L(x, y, y^*)$. In an exact learning scenario, the goal is to find the parameters of a cost function $\mathcal{C}$ such that for every training example

(x, y^*), the loss of the minimum cost output $\hat{y}$ equals l_{best}, i.e., $L(x, \hat{y}, y^*) = l_{best}$, where $\hat{y} = \arg min_{y \in \mathcal{Y}_{\mathbf{H}}(x)} \mathcal{C}(x, y)$. In practice, when exact learning isn't possible, the goal is to find a cost function such that the average loss over the training data of the predicted output using the cost function is minimized.

Algorithm 2 Cost Function Learning via Cross Validation

Input: $\mathcal{D}$ = Training examples, $\mathcal{S}_o$ = Search space definition, L = Loss function, $\mathcal{A}$ = Search procedure, τ_{max} = search time bound
Output: $\mathcal{C}$, the cost function

Divide the training set $\mathcal{D}$ into k folds $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k$
// Learn k different heuristics $\mathcal{H}_1, \cdots, \mathcal{H}_k$
for $i = 1$ to k **do**
  $T_i = \cup_{j \neq i} \mathcal{D}_j$ *// training data for heuristic $\mathcal{H}_i$*
  $\mathcal{H}_i =$ **Learn-Heuristic**$(T_i, \mathcal{S}_o, L, \mathcal{A}, \tau_{max})$ *// heuristic learning via Algorithm 1*
end for
// Generate ranking examples for cost function training
Intialize the set of ranking examples $\mathcal{R} = \emptyset$
for $i = 1$ to k **do**
  for each training example $(x, y^*) \in \mathcal{D}_i$ **do**
    Generate outputs by running the search procedure $\mathcal{A}$ with heuristic $\mathcal{H}_i$ for time bound τ_{max}: $\mathcal{Y}_{\mathcal{H}_i}(x) =$ **Generate-Outputs**$(x, \mathcal{S}_o, \mathcal{A}, \mathcal{H}_i, \tau_{max})$
    Compute the set of best loss outputs: $\mathcal{Y}_{best} = \{y \in \mathcal{Y}_{\mathcal{H}_i}(x) | L(x, y, y^*) = l_{best}\}$, where $l_{best} = min_{y \in \mathcal{Y}_{\mathcal{H}_i}(x)} L(x, y, y^*)$
    for each pair of outputs $(y_{best}, y) \in \mathcal{Y}_{best} \times \mathcal{Y}_{\mathcal{H}_i}(x) \setminus \mathcal{Y}_{best}$ **do**
      Add ranking example $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$ to $\mathcal{R}$
    end for
  end for
end for
// Train cost function on all the ranking examples
$\mathcal{C} =$ **Rank-Learner**$(\mathcal{R})$
return learned cost function $\mathcal{C}$

We formulate the cost function training problem as an instance of *rank learning* problem (Agarwal & Roth, 2005). More specifically, we want all the best loss outputs in $\mathcal{Y}_{\mathcal{H}}(x)$ to be ranked better than all the non-best loss outputs according to our cost function, which is a bi-partite ranking problem. Let $\mathcal{Y}_{best}$ be the set of all best loss outputs from $\mathcal{Y}_{\mathcal{H}}(x)$, i.e., $\mathcal{Y}_{best} = \{y \in \mathcal{Y}_{\mathcal{H}}(x) | L(x, y, y^*) = l_{best}\}$. We generate one ranking example for every pair of outputs $(y_{best}, y) \in \mathcal{Y}_{best} \times \mathcal{Y}_{\mathcal{H}}(x) \setminus \mathcal{Y}_{best}$, requiring that $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$. If the search procedure was able to generate the target output $y*$ (i.e., $l_{best} = 0$), this is similar to the standard learning in CRFs and SVM-Struct, but results in a much simpler rank-learning problem (cost function needs to rank the correct output above *only* the incorrect outputs generated during search). When the set of best loss outputs $\mathcal{Y}_{best}$ is very large, bi-partite ranking may result in a highly over-constrained problem. In such cases, one could relax the problem by attempting to learn a cost function that ranks at least one output in $\mathcal{Y}_{best}$ higher than all the non-best loss outputs. This can be easily implemented in an online-learning

framework as follows. If there is an error (i.e., the best cost output according to the current weights $\hat{y} \notin \mathcal{Y}_{best}$), the weights are updated to ensure that the best cost output $\hat{y}_{best} \in \mathcal{Y}_{best}$ according to the current weights is ranked better than all the outputs in $\mathcal{Y}_{\mathcal{H}}(x) \setminus \mathcal{Y}_{best}$.

It is important to note that both in theory and practice, the distribution of outputs generated by the learned heuristic $\mathcal{H}$ on the testing data may be slightly different from the one on training data. Thus, if we train $\mathcal{C}$ on the training examples used to train $\mathcal{H}$, then $\mathcal{C}$ is not necessarily optimized for the test distribution. To mitigate this effect, we train our cost function via cross validation (see Algorithm 2) by training the cost function on the data, which was not used to train the heuristic. This training methodology is commonly used in Re-ranking style algorithms (Collins, 2000) among others.

Algorithm 2 describes our approach for cost function training via cross validation. There are four main steps in the algorithm. First, we divide the training data $\mathcal{D}$ into k folds. Second, we learn k different heuristics, where each heuristic $\mathcal{H}_i$ is learned using the data from all the folds excluding the i^{th} fold (Steps 3-6). Third, we generate ranking examples for cost function learning as described above using each heuristic $\mathcal{H}_i$ on the data it was not trained on (Steps 9-17). Finally, we give the aggregate set of ranking examples $\mathcal{R}$ to a rank learner (e.g., Perceptron, SVM-Rank) to learn the cost function $\mathcal{C}$ (Step 19).

3.4 Rank Learner

In this section, we describe the specifics of the rank learner that can be used to learn both the heuristic and cost functions from the aggregate sets of ranking examples produced by the above algorithms. We can use any off-the-shelf rank-learning algorithm (e.g., Perceptron, SVM-Rank) as our base learner to train the heuristic function from the set of ranking examples $\mathcal{R}$. In our specific implementation we employed the online Passive-Aggressive (PA) algorithm (Crammer, Dekel, Keshet, Shalev-Shwartz, & Singer, 2006) as our base learner. Training was conducted for 50 iterations in all of our experiments.

PA is an online large-margin algorithm, which makes several passes over the training examples $\mathcal{R}$, and updates the weights whenever it encounters a ranking error. Recall that each ranking example is of the form $\mathcal{H}(x, y_1) < \mathcal{H}(x, y_2)$ for heuristic training and $\mathcal{C}(x, y_1) < \mathcal{C}(x, y_2)$ for cost function training, where x is a structured input with target output y^*, y_1 and y_2 are potential outputs for x such that $L(x, y_1, y^*) < L(x, y_2, y^*)$. Let $\Delta{>}0$ be the difference between the losses of the two outputs involved in a ranking example. We experimented with PA variants that use margin scaling (margin scaled by Δ) and slack scaling (errors weighted by Δ) (Tsochantaridis, Joachims, Hofmann, & Altun, 2005). Since margin scaling performed slightly better than slack scaling, we report the results of the PA variant that employs margin scaling. Below we give the full details of the margin scaling update.

Let w_t be the current weights of the linear ranking function. If there is a ranking error when cycling through the training data, i.e., $w_t \cdot \Phi(x, y_2) - w_t \cdot \Phi(x, y_1) < \sqrt{\Delta}$, the new weights w_{t+1} that corrects the error can be obtained using the following equation.

$$w_{t+1} = w_t + \tau_t(\Phi(x, y_2) - \Phi(x, y_1))$$

where the learning rate τ_t is given by

$$\tau_t = \frac{w_t \cdot \Phi(x, y_1) - w_t \cdot \Phi(x, y_2) + \sqrt{\Delta}}{\|\Phi(x, y_2) - \Phi(x, y_1)\|^2}$$

This specific update has been previously used for cost-sensitive multiclass classification (Crammer et al., 2006) (See Equation 51) and for structured output problems (Keshet, Shalev-Shwartz, Singer, & Chazan, 2005) (See Equation 7).

4. Experiments and Results

In this section we empirically investigate our $\mathcal{HC}$-Search approach and compare it against the state-of-the-art in structured prediction.

4.1 Datasets

We evaluate our approach on the following four structured prediction problems including three benchmark sequence labeling problems and a 2D image labeling problem.

- **Handwriting Recognition (HW).** The input is a sequence of binary-segmented handwritten letters and the output is the corresponding character sequence $[a-z]^+$. This dataset contains roughly 6600 examples divided into 10 folds (Taskar et al., 2003). We consider two different variants of this task as in the work of Hal Daumé III, Langford, and Marcu (2009). In the `HW-Small` version, we use one fold for training and the remaining 9 folds for testing, and vice-versa in `HW-Large`.
- **NETtalk Stress.** This is a text-to-speech mapping problem, where the task is to assign one of the 5 stress labels to each letter of a word. There are 1000 training words and 1000 test words in the standard dataset. We use a sliding window of size 3 for observational features.
- **NETtalk Phoneme.** This is similar to NETtalk Stress except that the task is to assign one of the 51 phoneme labels to each letter of the word.
- **Scene labeling.** This data set contains 700 images of outdoor scenes (Vogel & Schiele, 2007). Each image is divided into patches by placing a regular grid of size 10×10 over the entire image, where each patch takes one of the 9 semantic labels (*sky, water, grass, trunks, foliage, field, rocks, flowers, sand*). Simple appearance features including color, texture and position are used to represent each patch. Training was performed with 600 images, and the remaining 100 images were used for testing.

4.2 Experimental Setup

For our $\mathcal{HC}$-Search experiments, we use the Limited Discrepancy Space (LDS) exactly as described in the work of Doppa et al. (2012) as our search space over structured outputs. Prior work with $\mathcal{HC}$-Search has shown that greedy search works quite well for most structured prediction tasks, particularly when using the LDS space (Doppa et al., 2012). Hence, we consider only greedy search in our experiments. We would like to point out that experiments not shown using beam search and best first search produce similar results. During

training and testing we set the search time bound τ to be 25 search steps for all domains except for scene labeling, which has a much larger search space and uses $\tau = 150$. We found that using values of τ larger than these did not produce noticeable improvement. For extremely small values of τ, performance tends to be worse, but it increases quickly as τ is made larger. We will also show results for the full spectrum of time bounds later. For all domains, we learn linear heuristic and cost functions over second order features unless otherwise noted. In this case, the feature vector measures features over neighboring label pairs and triples along with features of the structured input. We measure error with Hamming loss unless otherwise noted.

4.3 Comparison to State-of-the-Art

We compare the results of our **$\mathcal{HC}$-Search** approach with other structured prediction algorithms including **CRFs** (Lafferty et al., 2001), **SVM-Struct** (Tsochantaridis et al., 2004), **Searn** (Hal Daumé III et al., 2009), **Cascades** (Weiss & Taskar, 2010) and **$\mathcal{C}$-Search**, which is identical to $\mathcal{HC}$-Search except that it uses a single-function for output space search (Doppa et al., 2012). We also show the performance of **Recurrent**, which is a simple recurrent classifier trained exactly as in the work of Doppa et al. (2012). The top section of Table 1 shows the error rates of the different algorithms. For scene labeling it was not possible to run CRFs, SVM-Struct, and Cascades due to the complicated grid structure of the outputs (hence the '-' in the table). We report the best published results of CRFs, SVM-Struct, and Searn. Cascades was trained using the implementation (Weiss, 2014) provided by the authors, which can be used for sequence labeling problems with Hamming loss. We would like to point out that the results of cascades differ from those that appear in the work of Doppa, Fern, and Tadepalli (2013) and are obtained using an updated[2] version of cascades training code. Across all benchmarks, we see that results of $\mathcal{HC}$-Search are comparable or significantly better than the state-of-the-art including $\mathcal{C}$-Search, which uses a single function as both heuristic function and cost function. The results in the scene labeling domain are the most significant improving the error rate from 27.05 to 19.71. These results show that $\mathcal{HC}$-Search is a state-of-the-art approach across these problems and that learning separate heuristic and cost functions can significantly improve output-space search.

4.4 Higher-Order Features

One of the advantages of our approach compared to many frameworks for structured prediction is the ability to use more expressive feature spaces without paying a huge computational price. The bottom part of Table 1 shows results using third-order features (compared to second-order above) for $\mathcal{HC}$-Search, $\mathcal{C}$-Search and Cascades. Note that it is not practical to run the other methods using third-order features due to the substantial increase in inference time. The overall error of $\mathcal{HC}$-Search with higher-order features slightly improved compared to using second-order features across all benchmarks and is still better than the error-rates of $\mathcal{C}$-Search and Cascades with third-order features, with the exception of Cascades on `HW-Large`. In fact, $\mathcal{HC}$-Search using only second-order features is still outperforming the third-order results of the other methods on three out of five domains.

2. Personal communication with the author

Algorithms	Datasets				
	HW-Small	HW-Large	Stress	Phoneme	Scene labeling

a. Comparison to state-of-the-art

$\mathcal{HC}$-Search	**12.81**	03.23	**17.58**	**16.91**	**19.71**
$\mathcal{C}$-Search	17.03	07.16	21.07	20.81	27.05
CRF	19.97	13.11	21.48	21.09	-
SVM-Struct	19.64	12.49	22.01	21.70	-
Recurrent	34.33	25.13	27.18	26.42	43.36
Searn	17.88	09.42	23.85	22.74	37.69
Cascades	13.02	**03.22**	20.41	17.56	-

b. Results with Third-Order Features

$\mathcal{HC}$-Search	**10.04**	02.21	**16.32**	**14.29**	**18.25**
$\mathcal{C}$-Search	14.15	04.76	19.36	18.19	25.79
Cascades	10.82	**02.16**	19.51	17.41	-

Table 1: Error rates of different structured prediction algorithms.

4.5 Loss Decomposition Analysis

We now examine $\mathcal{HC}$-Search and $\mathcal{C}$-Search in terms of their loss decomposition (see Equation 4) into generation loss $\epsilon_{\mathcal{H}}$ and selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$. Both of these quantities can be easily measured for both $\mathcal{HC}$-Search and $\mathcal{C}$-Search by keeping track of the best loss output generated by the search (guided either by a heuristic or the cost function for $\mathcal{C}$-Search) across the testing examples. Table 2 shows these results, giving the overall error $\epsilon_{\mathcal{HC}}$ and its decomposition across our benchmarks for both $\mathcal{HC}$-Search and $\mathcal{C}$-Search.

We first see that generation loss $\epsilon_{\mathcal{H}}$ is very similar for $\mathcal{C}$-Search and $\mathcal{HC}$-Search across the benchmarks with the exception of scene labeling, where $\mathcal{HC}$-Search generates slightly better outputs. This shows that at least for the LDS search space the difference in performance between $\mathcal{C}$-Search and $\mathcal{HC}$-Search cannot be explained by $\mathcal{C}$-Search generating lower quality outputs. Rather, the difference between the two methods is most reflected by the difference in selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$, meaning that $\mathcal{C}$-Search is not as effective at ranking the outputs generated during search compared to $\mathcal{HC}$-Search. This result clearly shows the advantage of separating the roles of $\mathcal{C}$ and $\mathcal{H}$ and is understandable in light of the training mechanism for $\mathcal{C}$-Search. In that approach, the cost function is trained to satisfy constraints related to both the generation loss and selection loss. It turns out that there are many more generation loss constraints, which we hypothesize biases $\mathcal{C}$-Search toward low generation loss at the expense of selection loss.

These results also show that for both methods the selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$ contributes significantly more to the overall error compared to $\epsilon_{\mathcal{H}}$. This shows that both approaches are able to uncover very high-quality outputs, but are unable to correctly rank the generated outputs according to their losses. This suggests that a first avenue for improving the results of $\mathcal{HC}$-Search would be to improve the cost function learning component, e.g. by using non-linear cost functions.

4.6 Ablation Study

To futher demonstrate that having two separate functions (heuristic and cost function) as in $\mathcal{HC}$-Search will lead to more accurate predictions compared to using a single function as in $\mathcal{C}$-Search, we perform some ablation experiments. In this study, we take the learned heuristic function $\mathcal{H}$ and cost function $\mathcal{C}$ in the $\mathcal{HC}$-Search framwork, and use only one of them to make predictions. For example, $\mathcal{HH}$-Search corresponds to the configuration when we use the function $\mathcal{H}$ as both heuristic and cost function. Similarly, $\mathcal{CC}$-Search corresponds to the configuration when we use the function $\mathcal{C}$ as both heuristic and cost function.

Table 2b shows the results for these ablation experiments. We can make several interesting observations from these results. First, the overall error of $\mathcal{HC}$-Search is significantly better than that of $\mathcal{HH}$-Search and $\mathcal{CC}$-Search. Second, the selection loss for $\mathcal{HH}$-Search increases compared to that of $\mathcal{HC}$-Search. This is understandable because $\mathcal{H}$ is not trained to score the candidate outputs that are generated during search. Third, the generation loss for $\mathcal{CC}$-Search increases compared to that of $\mathcal{HC}$-Search and this behavior is significant (increases to 11.24 compared to 5.82) for the scene labeling task. All these results provide further evidence for the importance of separating the training of the heuristic and cost functions.

Datasets	HW-Small			HW-Large			Stress			Phoneme			Scene		
Error	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$
a. $\mathcal{HC}$-Search vs. $\mathcal{C}$-Search															
$\mathcal{HC}$-Search	12.8	04.7	08.0	03.2	00.7	02.7	17.5	02.7	14.7	16.9	03.4	13.4	19.7	05.8	13.8
$\mathcal{C}$-Search	17.5	04.9	12.6	07.1	00.9	06.2	21.0	03.0	18.0	20.8	04.1	16.6	27.0	07.8	19.2
b. Results for Ablation study															
$\mathcal{HH}$-Search	18.4	04.7	13.7	07.9	00.7	7.2	22.5	02.7	19.7	22.1	03.4	18.7	32.1	07.8	24.3
$\mathcal{CC}$-Search	16.2	05.3	10.9	06.6	01.7	04.9	19.1	03.2	15.8	21.6	04.3	17.3	25.3	11.2	14.0
c. Results with heuristic function training via DAgger															
$\mathcal{HC}$-Search	12.0	03.9	08.1	03.1	00.4	02.6	17.2	02.2	15.0	16.8	03.0	13.8	18.0	03.7	14.3
$\mathcal{C}$-Search	15.1	04.6	09.9	05.1	00.8	03.6	20.3	02.8	17.1	19.0	03.9	14.7	24.2	05.9	18.3
d. Results with Oracle Heuristic															
$\mathcal{LC}$-Search (Oracle $\mathcal{H}$)	10.1	00.2	09.9	03.0	00.5	02.5	14.1	00.2	13.9	12.2	00.5	11.7	16.3	00.3	16.0

Table 2: $\mathcal{HC}$-Search: Error decomposition of heuristic and cost function.

4.7 Results for Heuristic Training via DAgger

Our heuristic learning approach follows the simplest approach to imitation learning, exact imitation, where the learner attempts to exactly imitate the observed expert trajectories (here imitate search with the oracle heuristic). While our experiments show that exact imitation performs quite well, it is known that exact imitation has certain deficiencies in general. In particular, functions trained via exact imitation can be prone to error propagation (Kääriäinen, 2006; Ross & Bagnell, 2010), where errors made at test time change the distribution of decisions encountered in the future compared to the training distribution. To address this problem, more sophisticated imitation learning algorithms have been developed, with a state-of-the-art approach being DAgger (Ross, Gordon, & Bagnell, 2011).

Here we consider whether DAgger can improve our heuristic learning and in turn overall accuarcy.

DAgger is an iterative algorithm, where each iteration adds imitation data to an aggregated data set. The first iteration follows the exact imitation approach, where data are collected by observing an expert trajectory (or a number of them). After each iteration an imitation function (here a heuristic) is learned from the current data. Successive iterations generate trajectories by following a mixture of expert suggestions (in our case ranking decisions) and suggestions of the most recently learned imitation function. Each decision point along the trajectory is added to the aggregate data set by labeling it by the expert decision. In this way, later iterations allow DAgger to learn from states visited by its possibly erroneous learned functions and correct its mistakes using the expert input. Ross et al. (2011) show that during the iterations of DAgger just using the learned policy without mixing the expert policy performs very well across diverse domains. Therefore, we use the same approach in our DAgger experiments. In our experiments we run 5 iterations of DAgger, noting that no noticable improvement was observed after 5 iterations.

Table 2c shows the results of $\mathcal{HC}$-Search and $\mathcal{C}$-Search obtained by training with DAgger. For $\mathcal{HC}$-Search, the generation loss ($\epsilon_{\mathcal{H}}$) improved slightly on the sequence labeling problems as there is little room for improvement, but DAgger leads to significant improvement in the generation loss on the more challenging problem of scene labeling. We can also see that the overall error of $\mathcal{HC}$-Search for scene labeling reduces due to improvement in generation loss showing that cost function is able to leverage the better outputs produced by the heuristic. Similarly, the overall error of $\mathcal{C}$-Search also improved with DAgger across the board and we see most significant improvements for handwiriting and scene labeling domains. It is interesting to note that unlike $\mathcal{HC}$-Search, the improvement in $\mathcal{C}$-Search is mostly due the improvement in the selection loss ($\epsilon_{\mathcal{C}|\mathcal{H}}$) except for scene labeling task, where it is due to the improvement in both generation loss and selection loss.

These results show that improving the heuristic learning is able to improve overall performance. What is not clear is whether further improvement, perhaps due to future advances in imitation learning, would yet again lead to overall improvement. That is, while it may be possible to further improve the generation loss, it is not clear that the cost function will be able to exploit such improvments. To help evaluate this we ran an experiment where we gave $\mathcal{HC}$-Search the true loss function to use as a heuristic (an oracle heuristic), i.e., $\mathcal{H}(x, y) = L(x, y, y^*)$, during both training of the cost function and testing. This provides an assessment of how much better we might be able to do if we could improve heuristic learning. The results in Table 2, which we label as LC-Search (Oracle $\mathcal{H}$) show that when using the oracle heuristic, $\epsilon_{\mathcal{H}}$ is negligible as we might expect and smaller than observed for $\mathcal{HC}$-Search. This shows that it may be possible to further improve our heuristic learning via better imitation.

We also see from the oracle results that the overall error $\epsilon_{\mathcal{HC}}$ is better than that of $\mathcal{HC}$-Search, but for HW-Small and Scene labeling tasks, the selection error $\epsilon_{\mathcal{C}|\mathcal{H}}$ got slightly worse.. This indicates that our cost function learner is able to leverage, to varying degrees, the better outputs produced by the oracle heuristic. This suggests that improving the heuristic learner in order to reduce the generation loss could be a viable way of further reducing the overall loss of $\mathcal{HC}$-Search, even without altering the current cost learner. However, as we saw above there is much less room to improve the heuristic learner for these

data sets and hence the potential gains are less than for directly trying to improve the cost learner.

4.8 Results for Training with Different Time bounds

We also trained $\mathcal{HC}$-Search for different time bounds (i.e., number of greedy search steps) to see how the overall loss, generation loss and selection loss vary as we increase the training time bound. In general, as the time bound increases, the generation loss will monotonically decrease, since strictly more outputs will be encountered. On the other hand the difficulty of cost function learning can increase as the time bound grows since it must learn to distinguish between a larger set of candidate outputs. Thus, the degree to which the overall error decreases (or grows) with the time bound depends on a combination of how much the generation loss decreases and whether the cost function learner is able to accurately distinguish improved outputs.

Figure 5 shows the performance of $\mathcal{HC}$-Search for the full spectrum of time bounds. Qualitatively, we see that the generation loss, due to the heuristic, decreases remarkably fast and for most benchmarks improves very little after the initial decrease. We also see that the cost function learner achieves a relatively stable selection loss in a short time, though it does increase a bit with time in most cases. The combined effect is that we see the overall error $\epsilon_{\mathcal{HC}}$ improves quickly as we increase the time bound and the improvement tends to be very small beyond certain time bound. Also, in some cases (e.g., phoneme prediction and scene labeling) performance tends to get slightly worse for very large time bounds, which happens when the increase in selection loss is not counteracted by a decreased generation loss.

		Test	
	Loss Function	`Hamming`	`VC`
Train	`Hamming`	1757	4658
	`VC`	1769	4620

Table 3: Results for training with non-hamming loss functions.

4.9 Results for Training with Non-Hamming Loss functions

One of the advantages of $\mathcal{HC}$-Search compared to many other approaches for structured prediction is that it is sensitive to the loss function used for training. So we trained $\mathcal{HC}$-Search with different loss functions on the handwriting domain to verify if this is true in practice or not. We used hamming loss (uniform misclassification cost of 1 for all characters) and Vowel-Consonant (VC) loss (different misclassification costs for vowels and consonants) for this experiment. For VC loss, we used misclassification costs of 4 and 2 for vowels and consonants respectively. Training was done on 5 folds and the remaining 5 folds were used for testing. Table 4.8 shows the results for training and testing with the two loss functions. We report cumulative loss over all the testing examples. As we can see, for any testing loss function, training with the same loss function gives slightly better performance than training using a different loss function. This shows that our $\mathcal{HC}$-Search learning approach

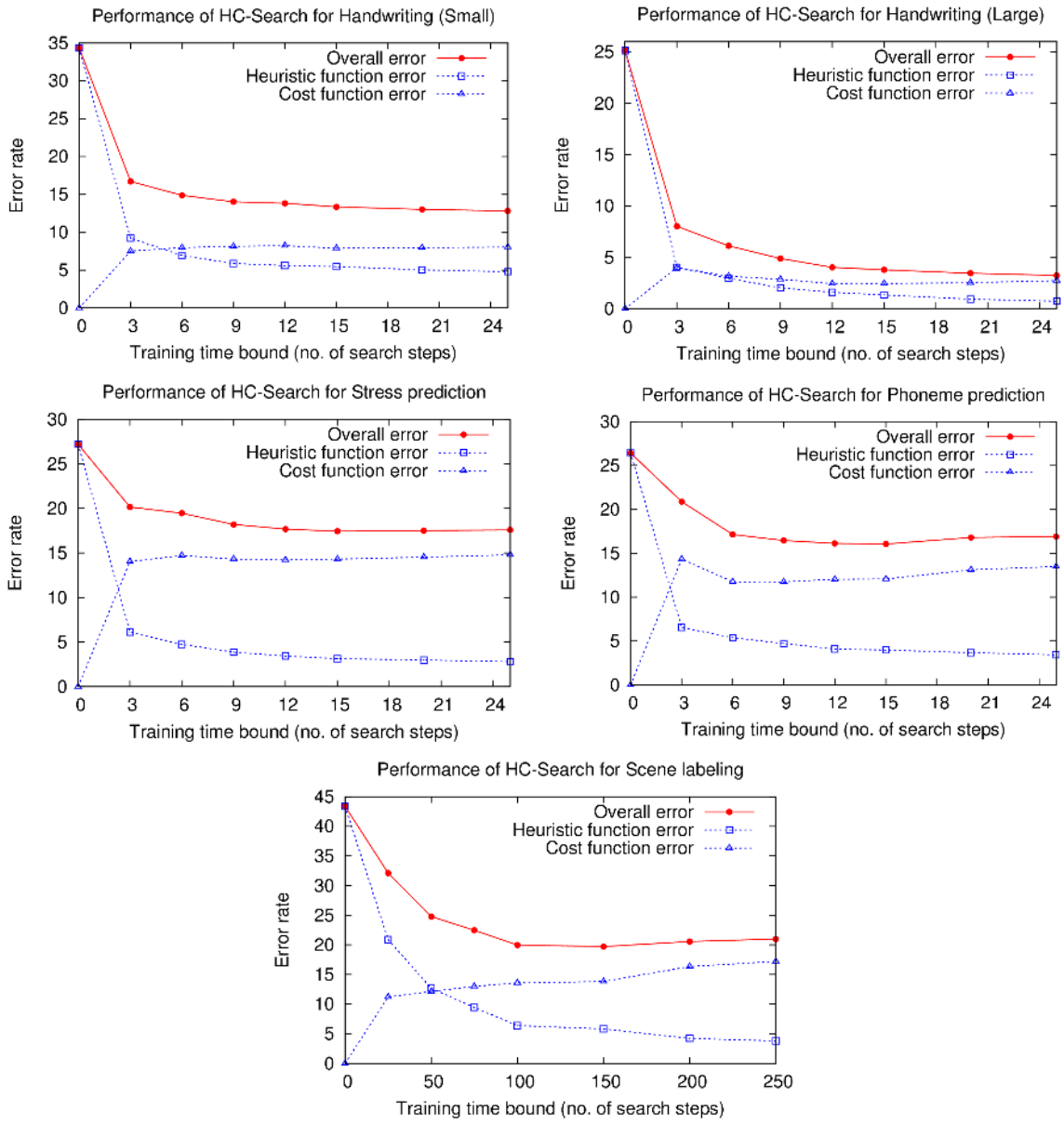

Figure 5: $\mathcal{HC}$-Search results for training with different time bounds. We have training time bound (i.e., no. of greedy search steps) on x-axis and error on y-axis. There are three curves in each graph corresponding to overall loss $\epsilon_{\mathcal{HC}}$, generation loss $\epsilon_{\mathcal{H}}$ and selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$.

is sensitive to the loss function. However, this result may not hold generally and very much depends on the problem structure, loss function and the ability of our cost function to capture that loss.

4.10 Discussion on Efficiency of the $\mathcal{HC}$-Search Approach

In our $\mathcal{HC}$-Search framework, the basic computational elements include generating candidate states for a given state; computing the heuristic function features via $\Phi_{\mathcal{H}}$ and cost function features via $\Phi_{\mathcal{C}}$ for all the candidate states; and computing the heuristic and cost scores via the learned heuristic and cost function pair $(\mathcal{H}, \mathcal{C})$. The computational time for generating the candidate states depends on the employed search space $\mathcal{S}_o = (I, S)$, where I is the initial state function and S is the successor function. For example, the generation of candidates will be very efficient with Flipbit space compared to the LDS space (involves running the recurrent classifier for every action specified by the successor function S). Therefore, the efficiency of the overall approach depends on the size of the candidate set and can be greatly improved by generating fewer candidate states (e.g., via pruning) or parallelizing the computation. We have done some preliminary work in this direction by introducing sparse versions of both LDS and Flipbit search spaces by pruning actions based on the recurrent classifier scores (as specified by the prunining parameter k). This simple pruning strategy resulted in 10-fold speedup with little or no loss in accuracy across several benchmark problems (Doppa et al., 2014a). However, more work needs to be done on learning pruning rules to improve the efficiency of the $\mathcal{HC}$-Search approach.

5. Engineering Methodology for Applying $\mathcal{HC}$-Search

In this section, we describe an engineering methodology for applying our $\mathcal{HC}$-Search framework to new problems. At a very high-level, the methodology involves selecting an effective time-bounded search architecture (search space, search procedure, and search time-bound), and leveraging the loss decomposition in terms of generation and selection loss for training and debugging the heuristic and cost functions. Below we describe these steps in detail.

5.1 Selection of Time-bounded Search Architecture

A time-bounded search architecture can be instantiated by selecting a search space, search strategy, and search time-bound. As we mentioned before, the effectiveness of $\mathcal{HC}$-Search depends critically on the quality of the search space (i.e., search depth at which target outputs can be found) that is being employed. In fact, our prior work empirically demonstrated that the performance gap of the search architectures with Flipbit space and LDS space grows as the difference between their target depths increase (Doppa et al., 2014a). Therefore, it is important to select/design a high-quality search space for the problem at hand.

If there exists a greedy predictor for the structured prediction problem, one could leverage it to define an appropriate variant of the LDS space. Fortunately, there are greedy predictors for several problems in natural language processing, computer vision, relational networks, and planning with preferences. For example, transition-based parsers for dependency parsing (Nivre, 2008; Goldberg & Elhadad, 2010); greedy classifiers for co-reference

resolution (Chang, Samdani, & Roth, 2013; Stoyanov & Eisner, 2012) and event extraction (Li, Ji, & Huang, 2013); sequential labelers for boundary detection of objects in images (Payet & Todorovic, 2013); iterative classifiers for collective inference in relational networks (Sen, Namata, Bilgic, Getoor, Gallagher, & Eliassi-Rad, 2008; Doppa, Yu, Tadepalli, & Getoor, 2009, 2010); classifier chains for multi-label prediction (Read, Pfahringer, Holmes, & Frank, 2011); and greedy planners for planning with preferences (Xu, Fern, & Yoon, 2010). In general, designing high-quality search spaces is a key research topic and more work needs to be done in this direction. Learning search operators ("macro actions") or transformation rules as in Transformation-based Learning (TBL) (Brill, 1995) to optimize the search space is one of the many possibilities. Sometimes problem structure can also help in designing effective search spaces. For example, in most multi-label prediction problems, the outputs which are binary vectors have a small number of active labels (highly sparse). So a simple flipbit space initialized with the null vector can be very effective (Doppa, Yu, Ma, Fern, & Tadepalli, 2014b).

After picking the search space, we need to select an appropriate search procedure and search time-bound. The effectiveness of a search architecture can be measured by performing oracle search (true loss function used as both heuristic and cost function) on the training data. So one could perform oracle search (LL-Search) with different search procedures (e.g., greedy and beam search) for different time-bounds and select the search procedure that is more effective. We did not see benefit with beam search for the problems we considered, but we expect that this can change for harder problems with non-Hamming loss functions (e.g., B-Cubed score for co-reference resolution). If the search space is not redundant, then we can fix the search time-bound to a value where the performance of the search architecture stagnates. Otherwise, one should allow some slack so that the search procedure can recover from errors. In our experiments, we found that T (size of the structured output) is a reasonable value for the time-bound (Figure 5 provides justification for this choice).

5.2 Training and Debugging

The training procedure involves learning the heuristic $\mathcal{H}$ and cost function $\mathcal{C}$ to optimize the performance of the selected time-bounded search architecture on the training data. Following our staged learning approach, one could start with learning a heuristic via exact imitation of the oracle search. After that, the learned heuristic $\mathcal{H}$ should be evaluated by measuring the generation loss ($\mathcal{H}L$-Search configuration). If the performance of the $\mathcal{H}L$-Search configuration is acceptable with respect to the performance of LL-Search, we can move to cost function learning part. Otherwise, we can try to improve the heuristic by either employing more sophisticated imitation learning algorithms (e.g., DAgger), enriching the feature function $\Phi_{\mathcal{H}}$, or employing a more powerful rank learner. Similarly, after learning the cost function $\mathcal{C}$ conditioned on the learned heuristic, we can measure the selection loss. If the selection loss is very high, we can try to improve the cost function by either adding expressive features to $\Phi_{\mathcal{C}}$ or employing a more powerful rank learner.

6. Comparison to Related Work

As described earlier, the majority of structured prediction work has focused on the use of exact inference for computing outputs when it is tractable, and approximate inference

techniques, such as loopy belief propagation and relaxation methods, when it is not. Learning then is focused on tuning the cost function parameters in order to optimize various objective functions, which differ among learning algorithms (Lafferty et al., 2001; Taskar et al., 2003; Tsochantaridis et al., 2004; McAllester, Hazan, & Keshet, 2010). There are also approximate cost function learning approaches that do not employ any inference routine during training. For example, piece-wise training (Sutton & McCallum, 2009), Decomposed Learning (Samdani & Roth, 2012) and its special case pseudo-max training (Sontag, Meshi, Jaakkola, & Globerson, 2010) fall under this category. These training approaches are very efficient, but they still need an inference algorithm to make predictions during testing. In these cases, one could employ the Constrained Conditional Models (CCM) framework (Chang, Ratinov, & Roth, 2012) with some declarative (global) constraints to make predictions using the learned cost function. The CCM framework relies on the Integer Linear Programming (ILP) inference method (Roth & tau Yih, 2005). More recent work has attempted to integrate (approximate) inference and cost function learning in a principled manner (Meshi, Sontag, Jaakkola, & Globerson, 2010; Stoyanov, Ropson, & Eisner, 2011; Hazan & Urtasun, 2012; Domke, 2013). Researchers have also worked on using higher-order features for CRFs in the context of sequence labeling under the pattern sparsity assumption (Ye, Lee, Chieu, & Wu, 2009; Qian, Jiang, Zhang, Huang, & Wu, 2009). However, these approaches are not applicable for the graphical models where the sparsity assumption does not hold.

An alternative approach to addressing inference complexity is cascade training (Felzenszwalb & McAllester, 2007; Weiss & Taskar, 2010; Weiss, Sapp, & Taskar, 2010), where efficient inference is achieved by performing multiple runs of inference from a coarse level to a fine level of abstraction. While such approaches have shown good success, they place some restrictions on the form of the cost functions to facilitate cascading. Another potential drawback of cascades and most other approaches is that they either ignore the loss function of a problem (e.g. by assuming Hamming loss) or require that the loss function be decomposable in a way that supports loss augmented inference. Our approach is sensitive to the loss function and makes minimal assumptions about it, requiring only that we have a blackbox that can evaluate it for any potential output.

Classifier-based structured prediction algorithms avoid directly solving the Argmin problem by assuming that structured outputs can be generated by making a series of discrete decisions. The approach then attempts to learn a *recurrent classifier* that given an input $\mathbf{x}$ is iteratively applied in order to generate the series of decisions for producing the target output $\mathbf{y}$. Simple training methods (e.g. Dietterich, Hild, & Bakiri, 1995) have shown good success and there are some positive theoretical guarantees (Syed & Schapire, 2010; Ross & Bagnell, 2010). However, recurrent classifiers can be prone to error propagation (Kääriäinen, 2006; Ross & Bagnell, 2010). Recent work, e.g. SEARN (Hal Daumé III et al., 2009), SMiLe (Ross & Bagnell, 2010), and DAgger (Ross et al., 2011), attempts to address this issue using more sophisticated training techniques and have shown state-of-the-art structured-prediction results. However, all these approaches use classifiers to produce structured outputs through a single sequence of greedy decisions. Unfortunately, in many problems, some decisions are difficult to predict by a greedy classifier, but are crucial for good performance. In contrast, our approach leverages recurrent classifiers to define good

quality search spaces over complete outputs, which allows decision making by comparing multiple complete outputs and choosing the best.

There are also non-greedy methods that learn a scoring function to search in the space of partial structured outputs (DauméIII & Marcu, 2005; Daumé III, 2006; Xu, Fern, & Yoon, 2009b; Huang, Fayong, & Guo, 2012; Yu, Huang, Mi, & Zhao, 2013). All these methods perform online training, and differ only in the way search errors are defined and how the weights are updated when errors occur. Unfortunately, training the scoring function can be difficult because it is hard to evaluate states with partial outputs and the theoretical guarantees for the learned scoring function (e.g., convergence and generalization results) rely on strong assumptions (Xu et al., 2009b).

Our work is most closely related to the output space search approaches (Doppa et al., 2012; Wick et al., 2011), which use a single cost function to serve as both search heuristic and also to score the candidate outputs. Serving these dual roles often means that the cost function needs to make unclear tradeoffs, increasing the difficulty of learning. Our $\mathcal{HC}$-Search approach overcomes this deficiency by learning two different functions, a heuristic function to guide the search to generate high-quality candidate outputs, and a cost function to rank the candidate outputs. Additionally, the error decomposition of $\mathcal{HC}$-Search in terms of heuristic error and cost function error allows the human designers of the learning system to diagnose failures and take corrective measures.

Our approach is also related to Re-Ranking (Collins, 2002), which uses a generative model to propose a k-best list of outputs, which are then ranked by a separate ranking function. In contrast, rather than restricting to a generative model for producing potential outputs, our approach leverages generic search over efficient search spaces guided by a learned heuristic function that has minimal representational restrictions, and employs a learned cost function to rank the candidate outputs. Recent work on generating multiple diverse solutions in a probabilistic framework can be considered as another way of producing candidate outputs. A representative set of approaches in this line of work are diverse M-best (Batra, Yadollahpour, Guzmán-Rivera, & Shakhnarovich, 2012), M-best modes (Park & Ramanan, 2011; Chen, Kolmogorov, Zhu, Metaxas, & Lampert, 2013) and Determinantal Point Processes (Kulesza & Taskar, 2012).

The general area of speedup learning studied in the planning and search community is also related to our work (Fern, 2010). In these problems, the cost function is typically known and the objective is to learn control knowledge (i.e., heuristic function) for directing a search algorithm to a low-cost terminal node in the search space. For example, STAGE (Boyan & Moore, 2000) learns an evaluation function over the states to improve the performance of search, where value of a state corresponds to the performance of a local search algorithm starting from that state, (Zhang & Dietterich, 1995) use Reinforcement Learning (RL) methods to learn heuristics for job shop scheduling with the goal of minimizing the duration of the schedule. Unlike the problems in planning and combinatorial optimization, such a cost function is not given for the structured prediction problems. Therefore, our $\mathcal{HC}$-Search approach learns a cost function to score the structured outputs along with a heuristic function to guide the search towards low cost outputs.

7. Summary and Future Work

We introduced the $\mathcal{HC}$-Search framework for structured prediction whose principal feature is the separation of the cost function from search heuristic. We showed that our framework yields significantly superior performance to state-of-the-art results, and allows an informative error analysis and diagnostics.

Our investigation showed that the main source of error of existing output-space approaches including our own approach ($\mathcal{HC}$-Search) is the inability of cost function to correctly rank the candidate outputs produced by the output generation process. This analysis suggests that learning more powerful cost functions, e.g., Regression trees (Mohan, Chen, & Weinberger, 2011), with an eye towards anytime performance (Grubb & Bagnell, 2012; Xu, Weinberger, & Chapelle, 2012) would be productive. Our results also suggested that there is room to improve overall performance with better heuristic learning. Thus, another direction to pursue is heuristic function learning to speed up the process of generating high-quality outputs (Fern, 2010).

Future work includes applying this framework to more challenging problems in natural language processing (e.g., co-reference resolution, dependency parsing, and semantic parsing) and computer vision (e.g., object detection in biological images Lam, Doppa, Hu, Todorovic, Dietterich, Reft, & Daly, 2013, and multi-object tracking in complex sports videos Chen, Fern, & Todorovic, 2014). The effectiveness of $\mathcal{HC}$-Search approach depends on the quality of the search space, and therefore, more work needs to be done in learning to optimize search spaces by leveraging the problem structure. Similarly, studying pruning techniques to further improve the efficiency of both learning and inference is another useful direction.

Acknowledgements

The authors would like to thank the anonymous reviewers and Jason Eisner, the associate editor, for their comments and feedback. The first author would also like to thank Tom Dietterich for his encouragement and support throughout this work. This work was supported in part by NSF grants IIS 1219258, IIS 1018490 and in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-13-2-0033. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, the DARPA, the Air Force Research Laboratory (AFRL), or the US government. A preliminary version of this article was published at AAAI-2013 (Doppa et al., 2013)

Appendix A. Limited Discrepancy Search (LDS) Space

The Limited Discrepancy Search (LDS) space (Doppa et al., 2012, 2014a) is defined in terms of a learned recurrent classifier h. Thus, we start by describing recurrent classifier and then explain the key idea behind LDS space. For simplicity, we explain the main ideas using a sequence labeling problem (handwriting recognition task) noting that they generalize to non-sequence labeling problems (for full details see Doppa et al., 2012, 2014a).

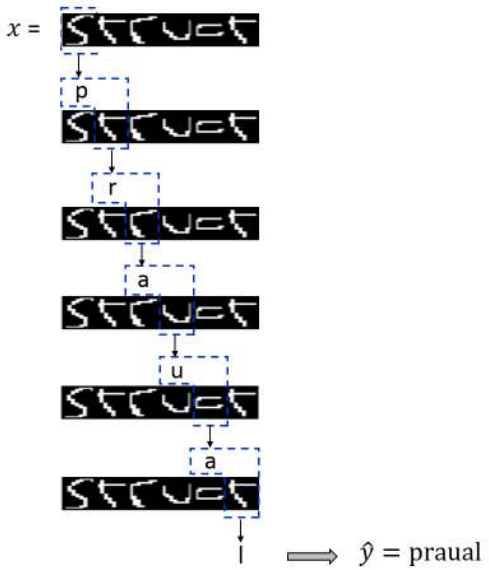

Figure 6: Illustration of recurrent classifier for handwriting recognition problem. The classifier predicts the labels in a left-to-right order. It makes the labeling decision at each position greedily based on the character image and the predicted label at previous position (shown by the dotted box). In this particular example, the classifier makes a mistake at the first position and this error propagates to other positions leading to a very bad output.

A.1 Recurrent Classifier

In a sequence labeling problem, the recurrent classifier produces a label at each position in sequence, based on an input in that position and the predicted labels at previous positions (Dietterich et al., 1995). If the learned classifier is accurate, then the number of incorrect labeling decisions will be relatively small. However, even a small number of errors can propagate and cause poor outputs.

Figure 6 illustrates recurrent classifier for a handwriting recognition example. The classifier predicts the labels in a left-to-right order. It makes the labeling decision at each position greedily based on the character image and the predicted label at the previous position (shown by the dotted box). In this particular example, the classifier makes a mistake at the first position and this error propagates leading to a very bad output (5 errors).

A.2 Limited Discrepancy Search (LDS)

LDS was originally introduced in the context of problem solving using heuristic search (Harvey & Ginsberg, 1995). The key idea behind LDS is to realize that if the classifier prediction was corrected at a small number of critical errors, then a much better output

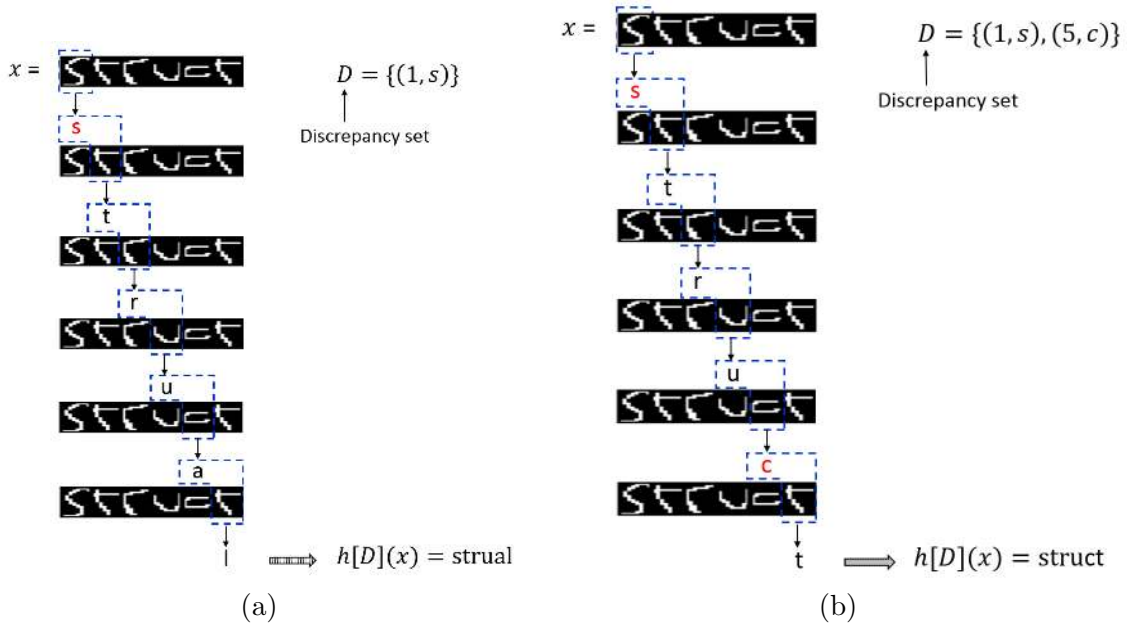

Figure 7: Illustration of Limited Discrepancy Search (LDS) for handwriting recognition problem. For any given discrepancy set D, we can generate a unique output by running the recurrent classifier with the changes from D. (a) LDS with one discrepancy. If introduce a discrepancy at the first position with label `s` (shown in red) and run the classifier, it is able to correct the two subsequent labels. (b) LDS with two discrepancies. If we introduce an additional discrepancy at fifth position with label `c` (shown in red) and run the classifier, we recover the target output `struct`.

would be produced. LDS conducts a (shallow) search in the space of possible corrections in the hope of finding an output better than the original.

Given a classifier h and a sequence of length T, a discrepancy is a pair (i, l) where $i \in \{1, \ldots, T\}$ is the index of sequence position and l is a label, which generally is different from the prediction of the classifier at position i. For any set of discrepancies D, we can generate a unique output $h[D](x)$ by running the classifier with changes in D. The discrepancies in D can be viewed as overriding the prediction of h at particular positions, possibly correcting for errors, or introducing new errors. At one extreme, when D is empty, we get the original output produced by the greedy classifier (see Figure 6). At the other extreme, when D specifies a label at each position, the output is not influenced by h at all and is completely specified by the discrepancy set. Figure 7 illustrates LDS for the same handwriting example. If we introduce a discrepancy at the first position with label `s` (shown in red) and run the classifier, it is able to correct the two subsequent labels (see Figure 7(a)). If we introduce an additional discrepancy at fifth position with label `c` (shown in red) and run the classifier, we recover the target output `struct` (see Figure 7(b)).

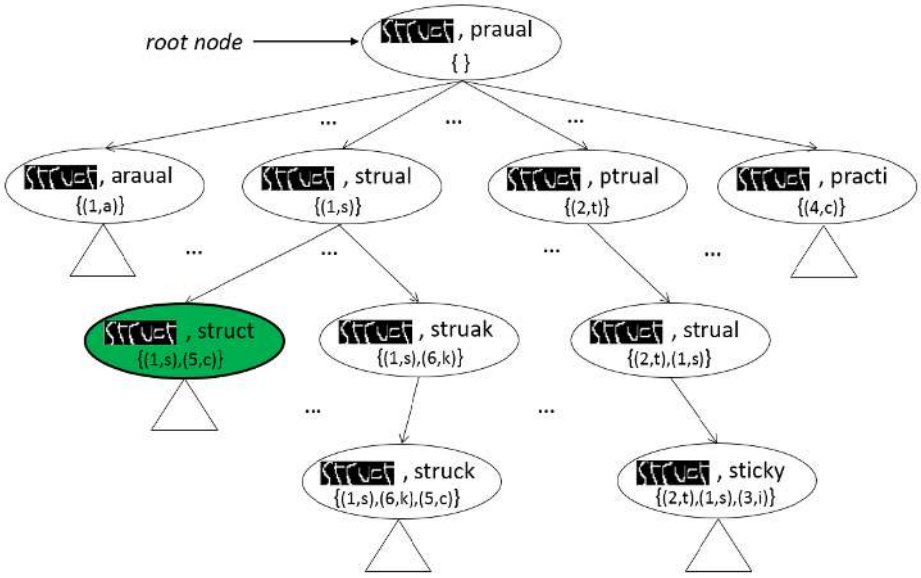

Figure 8: An example Limited Discrepancy Search (LDS) space for handwriting recognition problem. The highlighted state corresponds to the one with true output y^* at the smallest depth.

In practice, when h is reasonably accurate, we will be primarily interested in small discrepancy sets relative to the length of the sequence. The problem is that we do not know where the corrections should be made and thus LDS conducts a search over the discrepancy sets, usually from small to large sets.

A.3 LDS Space

Given a recurrent classifier h, we define the corresponding limited-discrepancy search space over complete outputs as follows. Each state in this search space is represented as (x, D) where x is a input sequence and D is a discrepancy set. We view a state (x, D) as equivalent to the input-output state $(x, h[D](x))$. The initial state function I simply returns $(x, \emptyset)$ which corresponds to the original output of the recurrent classifier. The successor function S for a state (x, D) returns the set of states of the form (x, D'), where D' is the same as D, but with an additional discrepancy. In this way, a path through the LDS search space starts at the output generated by the recurrent classifier and traverses a sequence of outputs that differ from the original by some number of discrepancies. Given a reasonably accurate h, we expect that high-quality outputs will be generated at relatively shallow depths of this search space and hence will be generated quickly.

Figure 8 illustrates[3] the limited-discrepancy search space. Each state consists of the input x, a discrepancy set D and the output produced by running the classifier with the specified discrepancy set, i.e., $h[D](x)$. The root node has an empty discrepancy set. Nodes at level one contain discrepancy sets of size one. The highlighted state corresponds to the smallest depth state containing the target output.

Appendix B. Hardness Proof for $\mathcal{HC}$-Search Consistency Problem

Theorem 2. *The $\mathcal{HC}$-Search Consistency Problem for greedy search and linear heuristic and cost functions is NP-Hard even when we restrict to problems for which all possible heuristic functions uncover a zero loss output.*

Proof. We reduce from the Minimum Disagreement problem for linear binary classifiers, which was proven to be NP-complete in the work of Hoffgen, Simon, and Horn (1995). In one statement of this problem we are given as input a set of N, p-dimensional vectors $T = \{x_1, \ldots, x_N\}$ and a positive integer k. The problem is to decide whether or not there is a p-dimensional real-valued weight vector w such that $w \cdot x_i < 0$ for at most k of the vectors.

We first sketch the high-level idea of the proof. Given an instance of Minimum Disagreement, we construct an $\mathcal{HC}$-Search consistency problem with only a single structured training example. The search space corresponding to the training example is designed such that there is a single node n^* that has a loss of zero and all other nodes have a loss of 1. Further for all linear heuristic functions all greedy search paths terminate at n^*, while generating some other set of nodes/outputs on the path there. The search space is designed such that each possible path from the initial node to n^* corresponds to selecting k or fewer vectors from T, which we will denote by T^-. By traversing the path, the set of nodes generated (and hence must be scored by $\mathcal{C}$), say $\mathcal{N}$, includes feature vectors corresponding to those in $T - T^-$ along with the negation of the feature vectors in T^-. We further define n^* to be assigned the zero vector, so that the cost of that node is 0 for any weight vector.

In order to achieve zero loss given the path in consideration, there must be a weight vector $w_{\mathcal{C}}$ such that $w_{\mathcal{C}} \cdot x \geq 0$ for all $x \in \mathcal{N}$. By our construction this is equivalent to $w_{\mathcal{C}} \cdot x < 0$ for $x \in T^-$. If this is possible then we have found a solution to the Minimum Disagreement problem since $|T^-| \leq k$. The remaining details show how to construct this space so that there is a setting of the heuristic weights that can generate paths corresponding to all possible T^- in a way that all paths end at n^*. For completeness we describe this construction below.

Each search node in the space other than n^* is a tuple (i, m, t) where $1 \leq i \leq N$, $0 \leq m \leq k$, and t is one of 5 node types from the set $\{d, s^+, s^-, x^+, x^-\}$. Here i should be viewed as indexing an example $x_i \in T$ and m effectively codes how many instances in T have been selected to be mistakes and hence put in T^-. Finally, t encodes the type of the search node with the following meanings which will become more clear during the construction: d (decision), s^+ (positive selection), s^- (negative selection), x^+ (positive instance), x^- (negative instance). The search space is constructed so that each example x_i

3. It may not be clear from this example, but we allow over-riding the discrepancies to provide the opportunity to recover from the search errors.

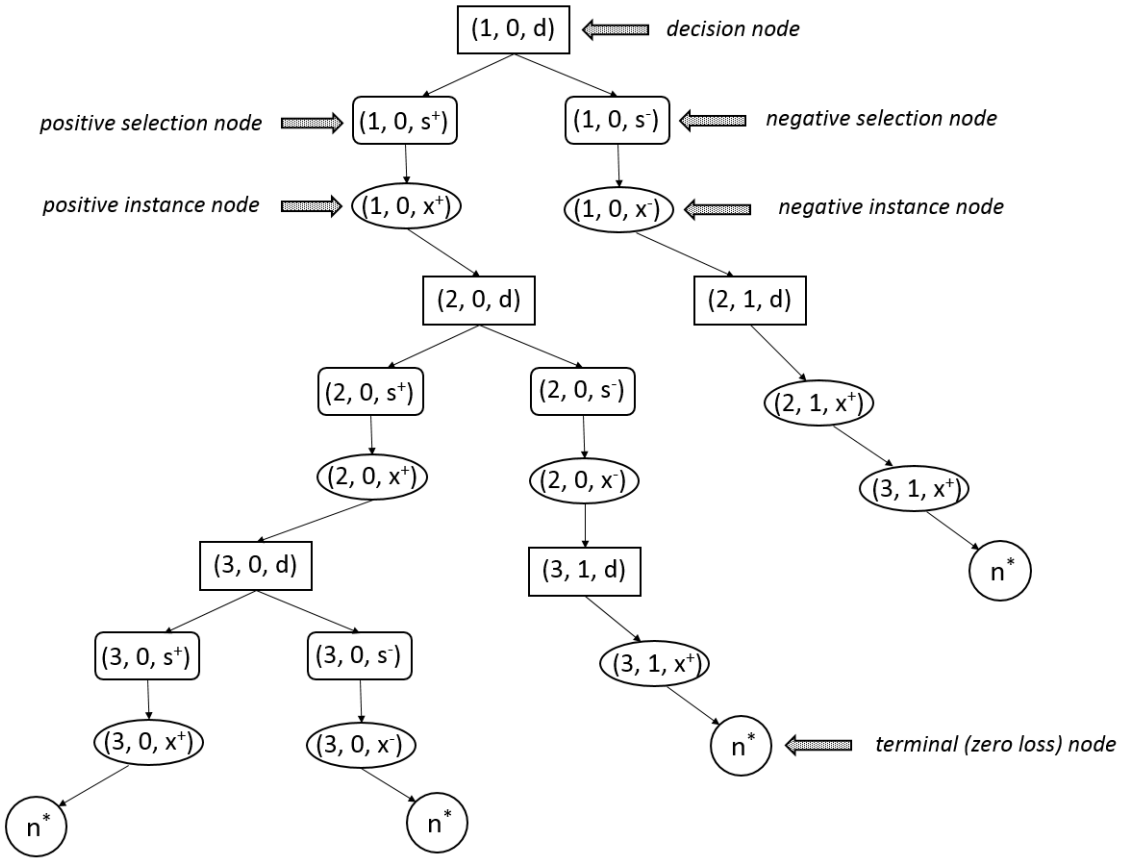

Figure 9: An example search space for $T = \{x_1, x_2, x_3\}$ and $k = 1$. All greedy paths terminate at the zero loss node n^* and no path selects more than one instance to include in the mistake set T^-.

is considered in order and a choice is made about whether to count it as a mistake (put it in T^-) or not. This choice is made at decision nodes, which all have the form (i, m, d), indicating that a decision is to be made about example i and that there have already been m examples selected for T^-. Each such decision node with $m < k$ has two children (i, m, s^-) and (i, m, s^+), which respectively correspond to selecting x_i to be in the mistake set or not. Later we will show how features are assigned to nodes so as to allow the heuristic to make any selection desired.

Each selection node has a single node as a child. In particular, a positive selection node (i, m, s^+) has the positive instance node (i, m, x^+) as a child, while negative selection nodes (i, m, s^-) has the negative instance node (i, m, x^-) as a child. Each such instance node effectively implements the process of putting x_i into T^- or not as will become clear when feature vectors are described below. After arriving at either a positive or negative instance node, the consideration of x_i is complete and we must move on to the decision for the next example x_{i+1}. Thus, a positive instance node (i, m, x^+) has the single child decision node

$(i+1, m, d)$, while a negative instance node has a single child decision node $(i+1, m+1, d)$, noting that the number of mistakes is incremented for negative nodes.

The final details of the search space structure ensure that no more than k mistakes are allowed and force all search paths to terminate at n^*. In particular, for any decision node (i, m, d) with $m = k$, we know that no more mistakes are allowed and hence no more decisions should be allowed. Thus, from any such node we form a path from it to n^* that goes through positive instance nodes $(i, m, x^+), \ldots, (N, m, x^+)$, which reflects that none of $\{x_i, \ldots, x_N\}$ will be in T^-. Figure 9 shows an example search space for our construction.

Given the above search space, which has polynomial size (since $k \leq N$), one can verify that for any set of k or fewer instances T^- there is a path from the root to n^* that goes through the negative instance nodes for instances in T^- and positive instance nodes for instances in $T - T^-$. Further, each possible path goes through either a positive or negative instance node for each instance and no more than k negative nodes. Thus there is a direct correspondence between paths and mistake sets T^-.

We now describe how to assign features to each node in a way that allows for the heuristic function to select each path and effectively construct the set T^-. For any node u the feature vector $\phi(u) = (x, s, b)$. The component x is an p-dimensional feature vector and will correspond to one of the x_i. The component s is an N-dimensional vector where $s_i \in \{-1, 1\}$ will implement the selection of instances. Finally b is a binary value that is equal to 1 for all non-instance nodes and is 0 for both positive and negative instance nodes. The mapping from nodes to feature vectors is as follows. Each decision node (i, m, d), is all zeros, except for $b = 1$. Each positive selection node (i, m, s^+) is all zeros except for $s_i = 1$ and $b = 1$. Negative selection nodes are similar except that $s_i = -1$. For a positive instance node (i, m, x^+) the feature vector is $(x_i, 0, 0)$ and for negative instance nodes (i, m, x^-) the feature vector is $(-x_i, 0, 0)$. Finally the feature vector for n^* is all zeros.

The key idea to note is that the heuristic function can effectively select a positive or negative selection node by setting the weight for s_i to be positive or negative respectively. In particular, the set of negative selection nodes visited (and hence negative instance nodes) correspond to the first k or fewer negative weight values for the s component of the feature vector. Thus, the heuristic can select any set of negative nodes that it wants to go through, but no more than k. On such a path there will be three types of nodes encountered that the cost function must rank. First, there will be control nodes (decision and selection nodes) that all have $b = 1$. Next there will be positive instance nodes that will have a feature vector $(x_i, 0, 0)$ and no more than k negative instance nodes with feature vectors $(-x_i, 0, 0)$. The cost function can easily rank n^* higher than the control nodes by setting the weight for b to be negative. Further if it can find heuristic weights for the x component that allows n^* to be ranked highest then that is a solution to the original minimum disagreement problem. Further if there is a solution to the disagreement problem it is easy to see that there will also be a solution to the $\mathcal{HC}$-Search consistency problem by selecting a heuristic that spans the proper set T^-. □

References

Agarwal, S., & Roth, D. (2005). Learnability of Bipartite Ranking Functions. In *Proceedings of International Conference on Learning Theory (COLT)*, pp. 16–31.

Batra, D., Yadollahpour, P., Guzmán-Rivera, A., & Shakhnarovich, G. (2012). Diverse M-Best Solutions in Markov Random Fields. In *Proceedings of European Conference on Computer Vision (ECCV)*, pp. 1–16.

Boyan, J. A., & Moore, A. W. (2000). Learning Evaluation Functions to Improve Optimization by Local Search. *Journal of Machine Learning Research (JMLR)*, *1*, 77–112.

Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, *21*(4), 543–565.

Chang, K.-W., Samdani, R., & Roth, D. (2013). A Constrained Latent Variable Model for Coreference Resolution. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 601–612.

Chang, M.-W., Ratinov, L.-A., & Roth, D. (2012). Structured Learning with Constrained Conditional Models. *Machine Learning Journal (MLJ)*, *88*(3), 399–431.

Chen, C., Kolmogorov, V., Zhu, Y., Metaxas, D., & Lampert, C. H. (2013). Computing the M Most Probable Modes of a Graphical Model. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Chen, S., Fern, A., & Todorovic, S. (2014). Multi-Object Tracking via Constrained Sequential Labeling. In *To appear in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

Collins, M. (2002). Ranking Algorithms for Named Entity Extraction: Boosting and the Voted Perceptron. In *ACL*.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research (JMLR)*, *7*, 551–585.

Daumé III, H. (2006). *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA.

DauméIII, H., & Marcu, D. (2005). Learning as Search Optimization: Approximate Large margin methods for Structured Prediction. In *ICML*.

Dietterich, T. G., Hild, H., & Bakiri, G. (1995). A Comparison of ID3 and Backpropagation for English Text-to-Speech Mapping. *Machine Learning Journal (MLJ)*, *18*(1), 51–80.

Domke, J. (2013). Structured Learning via Logistic Regression. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 647–655.

Doppa, J. R., Fern, A., & Tadepalli, P. (2012). Output Space Search for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*.

Doppa, J. R., Fern, A., & Tadepalli, P. (2013). HC-Search: Learning Heuristics and Cost Functions for Structured Prediction. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Doppa, J. R., Fern, A., & Tadepalli, P. (2014a). Structured Prediction via Output Space Search. *Journal of Machine Learning Research (JMLR)*, *15*, 1317–1350.

Doppa, J. R., Yu, J., Ma, C., Fern, A., & Tadepalli, P. (2014b). HC-Search for Multi-Label Prediction: An Empirical Study. In *To appear in Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Doppa, J. R., Yu, J., Tadepalli, P., & Getoor, L. (2009). Chance-Constrained Programs for Link Prediction. In *Proceedings of NIPS Workshop on Analyzing Networks and Learning with Graphs*.

Doppa, J. R., Yu, J., Tadepalli, P., & Getoor, L. (2010). Learning Algorithms for Link Prediction based on Chance Constraints. In *Proceedings of European Conference on Machine Learning (ECML)*, pp. 344–360.

Felzenszwalb, P. F., & McAllester, D. A. (2007). The Generalized A* Architecture. *Journal of Artificial Intelligence Research (JAIR)*, *29*, 153–190.

Fern, A. (2010). Speedup Learning. In *Encyclopedia of Machine Learning*, pp. 907–911.

Fern, A., Yoon, S. W., & Givan, R. (2006). Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)*, *25*, 75–118.

Goldberg, Y., & Elhadad, M. (2010). An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Proceedings of Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistic (HLT-NAACL)*, pp. 742–750.

Grubb, A., & Bagnell, D. (2012). SpeedBoost: Anytime Prediction with Uniform Near-Optimality. *Journal of Machine Learning Research - Proceedings Track*, *22*, 458–466.

Hal Daumé III, Langford, J., & Marcu, D. (2009). Search-based Structured Prediction. *Machine Learning Journal (MLJ)*, *75*(3), 297–325.

Harvey, W. D., & Ginsberg, M. L. (1995). Limited Discrepancy Search. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 607–615.

Hazan, T., & Urtasun, R. (2012). Efficient Learning of Structured Predictors in General Graphical Models. *CoRR*, *abs/1210.2346*.

Hoffgen, K.-U., Simon, H.-U., & Horn, K. S. V. (1995). Robust Trainability of Single Neurons. *Journal of Computer and System Sciences*, *50*(1), 114–125.

Huang, L., Fayong, S., & Guo, Y. (2012). Structured Perceptron with Inexact Search. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pp. 142–151.

Jiang, J., Teichert, A., Daumé III, H., & Eisner, J. (2012). Learned Prioritization for Trading Off Accuracy and Speed. In *Proceedings of Advances in Neural Information Processing (NIPS)*.

Kääriäinen, M. (2006). Lower Bounds for Reductions. In *Atomic Learning Workshop*.

Keshet, J., Shalev-Shwartz, S., Singer, Y., & Chazan, D. (2005). Phoneme Alignment based on Discriminative Learning. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, pp. 2961–2964.

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

Kulesza, A., & Taskar, B. (2012). Determinantal Point Processes for Machine Learning. *Foundations and Trends in Machine Learning*, *5*(2-3), 123–286.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of International Conference on Machine Learning (ICML)*, pp. 282–289.

Lam, M., Doppa, J. R., Hu, X., Todorovic, S., Dietterich, T., Reft, A., & Daly, M. (2013). Learning to Detect Basal Tubules of Nematocysts in SEM Images. In *ICCV Workshop on Computer Vision for Accelerated Biosciences (CVAB)*. IEEE.

Li, Q., Ji, H., & Huang, L. (2013). Joint Event Extraction via Structured Prediction with Global Features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 73–82.

McAllester, D. A., Hazan, T., & Keshet, J. (2010). Direct Loss Minimization for Structured Prediction. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1594–1602.

Meshi, O., Sontag, D., Jaakkola, T., & Globerson, A. (2010). Learning Efficiently with Approximate Inference via Dual Losses. In *Proceedings of International Conference on Machine Learning (ICML)*, pp. 783–790.

Mohan, A., Chen, Z., & Weinberger, K. Q. (2011). Web-Search Ranking with Initialized Gradient Boosted Regression trees. *Journal of Machine Learning Research - Proceedings Track*, *14*, 77–89.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, *34*(4), 513–553.

Park, D., & Ramanan, D. (2011). N-Best Maximal Decoders for Part Models. In *Proccedings of IEEE International Conference on Computer Vision (ICCV)*, pp. 2627–2634.

Payet, N., & Todorovic, S. (2013). SLEDGE: Sequential Labeling of Image Edges for Boundary Detection. *International Journal of Computer Vision (IJCV)*, *104*(1), 15–37.

Qian, X., Jiang, X., Zhang, Q., Huang, X., & Wu, L. (2009). Sparse Higher Order Conditional Random Fields for Improved Sequence Labeling. In *Proceedings of International Conference on Machine Learning (ICML)*.

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2011). Classifier Chains for Multi-Label Classification. *Machine Learning*, *85*(3), 333–359.

Ross, S., & Bagnell, D. (2010). Efficient Reductions for Imitation Learning. *Journal of Machine Learning Research - Proceedings Track*, *9*, 661–668.

Ross, S., Gordon, G. J., & Bagnell, D. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *Journal of Machine Learning Research - Proceedings Track*, *15*, 627–635.

Roth, D., & tau Yih, W. (2005). Integer Linear Programming Inference for Conditional Random Fields. In *Proceedings of International Conference on Machine Learning (ICML)*, pp. 736–743.

Samdani, R., & Roth, D. (2012). Efficient Decomposed Learning for Structured Prediction. In *Proceedings of International Conference on Machine Learning (ICML)*.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., & Eliassi-Rad, T. (2008). Collective Classification in Network Data. *AI Magazine*, *29*(3), 93–106.

Sontag, D., Meshi, O., Jaakkola, T., & Globerson, A. (2010). More data means less inference: A pseudo-max approach to structured learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2181–2189.

Stoyanov, V., & Eisner, J. (2012). Easy-first Coreference Resolution. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pp. 2519–2534.

Stoyanov, V., Ropson, A., & Eisner, J. (2011). Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 725–733.

Sutton, C. A., & McCallum, A. (2009). Piecewise Training for Structured Prediction. *Machine Learning Journal (MLJ)*, *77*(2-3), 165–194.

Syed, U., & Schapire, R. (2010). A Reduction from Apprenticeship Learning to Classification. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2253–2261.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-Margin Markov Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *Proceedings of International Conference on Machine Learning (ICML)*.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research (JMLR)*, *6*, 1453–1484.

Vogel, J., & Schiele, B. (2007). Semantic Modeling of Natural Scenes for Content-Based Image Retrieval. *International Journal of Computer Vision (IJCV)*, *72*(2), 133–157.

Weiss, D. (2014). Structured Prediction Cascades code. `http://code.google.com/p/structured-cascades/`.

Weiss, D., Sapp, B., & Taskar, B. (2010). Sidestepping Intractable Inference with Structured Ensemble Cascades. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2415–2423.

Weiss, D., & Taskar, B. (2010). Structured Prediction Cascades. *Journal of Machine Learning Research - Proceedings Track*, *9*, 916–923.

Wick, M. L., Rohanimanesh, K., Bellare, K., Culotta, A., & McCallum, A. (2011). SampleRank: Training Factor Graphs with Atomic Gradients. In *Proceedings of International Conference on Machine Learning (ICML)*.

Wick, M. L., Rohanimanesh, K., Singh, S., & McCallum, A. (2009). Training Factor Graphs with Reinforcement Learning for Efficient MAP Inference. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2044–2052.

Xu, Y., Fern, A., & Yoon, S. (2009a). Learning Linear Ranking Functions for Beam Search with Application to planning. *The Journal of Machine Learning Research*, *10*, 1571–1610.

Xu, Y., Fern, A., & Yoon, S. W. (2009b). Learning Linear Ranking Functions for Beam Search with Application to Planning. *Journal of Machine Learning Research (JMLR)*, *10*, 1571–1610.

Xu, Y., Fern, A., & Yoon, S. W. (2010). Iterative Learning of Weighted Rule Sets for Greedy Search. In *Proceedings of International Conference on Automated Planning and Systems (ICAPS)*, pp. 201–208.

Xu, Z., Weinberger, K., & Chapelle, O. (2012). The Greedy Miser: Learning under Test-time Budgets. In *Proceedings of International Conference on Machine Learning (ICML)*.

Ye, N., Lee, W. S., Chieu, H. L., & Wu, D. (2009). Conditional Random Fields with High-Order Features for Sequence Labeling. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 2196–2204.

Yu, H., Huang, L., Mi, H., & Zhao, K. (2013). Max-Violation Perceptron and Forced Decoding for Scalable MT Training. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1112–1123.

Zhang, W., & Dietterich, T. G. (1995). A Reinforcement Learning Approach to job-shop Scheduling. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1114–1120.