$\mathcal{HC}$-Search: Learning Heuristics and Cost Functions for Structured Prediction

Janardhan Rao Doppa and **Alan Fern** and **Prasad Tadepalli**
{doppa,afern,tadepall}@eecs.oregonstate.edu
School of EECS, Oregon State University, Corvallis, OR, USA, 97331

Abstract

Structured prediction is the problem of learning a function from structured inputs to structured outputs. 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 uses a separate learned cost function $\mathcal{C}$ to select a final prediction among those outputs. We can decompose the regret of the overall approach 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 regret in a greedy stagewise 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. Experiments on several benchmark domains show that our approach significantly outperforms the 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 tagging, the inputs and outputs are sequences of words and part of speech tags respectively. 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. Unfortunately exactly solving the Argmin problem is intractable except in limited cases such as when the dependency structure among features forms a tree (Lafferty, McCallum, and Pereira 2001; Taskar, Guestrin, and Koller 2003; Tsochantaridis et al. 2004). For more complex structures, heuristic inference methods such as loopy belief propagation and variational inference have shown some success in practice. However, the learning algorithms generally assume exact inference and their behavior in the context of heuristic inference is not well understood.

Another approach to dealing with the Argmin problem is to eliminate it altogether and instead attempt to learn a classifier that can greedily produce a structured output by making a series of discrete decisions (Dietterich, Hild, and Bakiri 1995; Hal Daumé III, Langford, and Marcu 2009; Ross, Gordon, and Bagnell 2011). Unfortunately, in many problems, some decisions are difficult to make by a greedy classifier, but are crucial for good performance. Cascade models (Felzenszwalb and McAllester 2007; Weiss and Taskar 2010; Weiss, Sapp, and Taskar 2010) achieve efficiency by performing inference on a sequence of models from coarse to finer levels. However, cascading places strong restrictions on the form of the cost functions.

We are inspired by the recent successes of output-space search approaches, which place few restrictions on the form of the cost function (Doppa, Fern, and Tadepalli 2012; Wick et al. 2011). These methods learn and use a cost function to direct a combinatorial search through a space of outputs, and return the least cost output uncovered. 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 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.

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 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. In addition to providing overall empirical results, we provide an in-depth analysis in terms of our loss decomposition that more precisely identifies the advantages over prior work and the reasons for our own failures.

Finally, we note that our methodology can be viewed as similar in spirit to Re-Ranking (Collins 2000), which uses a generative model to propose a k-best list of outputs, which are then ranked by a separate ranking function. In contrast, we search in efficient search spaces guided by a learned heuristic that has minimal restrictions on its representation.

2 $\mathcal{HC}$-Search

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 heuristics 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.

Search Spaces. Our approach is based on search in the space $\mathcal{S}_o$ of complete outputs, which we assume to be given. Every state in a search space over complete outputs consists of pairs of inputs and outputs (x, y), representing the possibility of predicting y as the output for 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 flip-bit space).

A variety of search spaces, such as the above flip-bit space, Limited Discrepancy Search (LDS) space (Doppa, Fern, and Tadepalli 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 experiments. The LDS space is defined in terms of a recurrent classifier 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 successors of (x, y) here consist of the results of running a recurrent classifier after changing one more label (introducing a discrepancy) in the current sequence y. In previous work, the LDS space is shown to be effective in uncovering high-quality outputs at relatively shallow search depths (Doppa, Fern, and Tadepalli 2012).

Our Approach. Our approach is parameterized by a search space, a heuristic search strategy $\mathcal{A}$ (e.g., greedy search), a learned heuristic function $\mathcal{H} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$, and a learned cost function $\mathcal{C} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$. Given an input x and a time bound τ, we generate an output by running $\mathcal{A}$ starting at $I(x)$ and guided by $\mathcal{H}$ until the time bound is exceeded. Each output uncovered by the search is scored according to $\mathcal{C}$ and we return the least-cost one as the final output.

More formally, let $\mathcal{Y}_{\mathcal{H}}(x)$ be the set of candidate outputs generated using heuristic $\mathcal{H}$ for a given input x. Further let $y^*_{\mathcal{H}}$ be the best loss output in this set, and $\hat{y}$ be the best cost output according to $\mathcal{C}$, i.e.,

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

Here, $y^*_{\mathcal{H}}$ is the best output that our approach could possibly return when using $\mathcal{H}$ and $\hat{y}$ is the output that it will actually return. 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

$$\mathcal{E}\left(\mathcal{H}, \mathcal{C}\right) = \mathbb{E}_{(x,y^*)\sim\mathcal{D}} \; L\left(x, \hat{y}, y^*\right) \qquad (1)$$

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

$$(\mathcal{H}^*, \mathcal{C}^*) = \arg min_{(\mathcal{H},\mathcal{C}) \in \mathbf{H} \times \mathbf{C}} \quad \mathcal{E}\left(\mathcal{H}, \mathcal{C}\right) \qquad (2)$$

Learning Complexity. We now consider the feasibility of an exact solution to this learning problem 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, and Yoon 2009). 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 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 good outputs to generate, and for only some of those will we be able to find an effective $\mathcal{C}$ to return the best one. We prove the following theorem.

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 all zero loss outputs.*

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

The reduction is relatively detailed and here we sketch the main idea. 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^*. □

Loss Decomposition. The above suggests that in general learning the optimal $(\mathcal{H}^*, \mathcal{C}^*)$ pair is impractical due to their potential interdependence. However, we observe a decomposition of the above error that leads to an effective training approach. In particular, the expected loss $\mathcal{E}(\mathcal{H}, \mathcal{C})$ can be decomposed into two parts: 1) the *generation loss* $\epsilon_\mathcal{H}$, due to the heuristic $\mathcal{H}$ not generating high-quality outputs, and 2) the *selection loss* $\epsilon_{\mathcal{C}|\mathcal{H}}$, the additional loss (conditional on $\mathcal{H}$) due to the cost function $\mathcal{C}$ not selecting the best loss output generated by the heuristic.

$$\mathcal{E}(\mathcal{H}, \mathcal{C}) = \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L(x, y^*_\mathcal{H}, y^*)}_{\epsilon_\mathcal{H}} + \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L(x, \hat{y}, y^*) - L(x, y^*_\mathcal{H}, y^*)}_{\epsilon_{\mathcal{C}|\mathcal{H}}} \quad (3)$$

As described in the next section, this decomposition allows us to specifically target our training to minimize each of these errors separately. In addition, as we show in our experiments, the terms of this decomposition can be easily measured for a learned $(\mathcal{H}, \mathcal{C})$ pair, which allows for an assessment of which function is more responsible for the loss.

In contrast to our approach, existing approaches for output space search (Doppa, Fern, and Tadepalli 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. It turns out that the expected loss of $\mathcal{HC}$-Search can be arbitrarily smaller than when restricting to using a single function $\mathcal{C}$ as the following proposition shows.

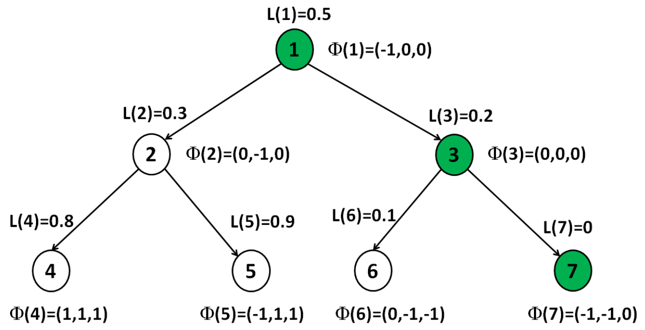

Figure 1: An example that illustrates that $\mathcal{C}$-Search can suffer arbitrarily large loss compared to $\mathcal{HC}$-Search.

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 1. We assume greedy search and linear $\mathcal{C}$ and $\mathcal{H}$ functions of features $\Phi(n)$. Node 7 is the least loss output. To find it, the heuristic $\mathcal{H}$ needs to satisfy, $\mathcal{H}(3)<\mathcal{H}(2)$, and $\mathcal{H}(7)<\mathcal{H}(6)$, which are easily satisfied. To return node 7 as the final output, the cost function must satisfy, $\mathcal{C}(7)<\mathcal{C}(1)$, $\mathcal{C}(7)<\mathcal{C}(2)$, $\mathcal{C}(7)<\mathcal{C}(3)$, and $\mathcal{C}(7)<\mathcal{C}(6)$. Again, it is possible to satisfy these, and $\mathcal{HC}$-Search can achieve zero loss on this problem. However, it can be verified that there is no single set of weights that simultaneously satisfies both sets of constraints ($\mathcal{C}(3)<\mathcal{C}(2)$ and $\mathcal{C}(7)<\mathcal{C}(1)$ in particular) as required by the $\mathcal{C}$-Search. By scaling the losses by constant factors we can make the loss suffered arbitrarily high. □

3 Learning Approach

Guided by the error decomposition in Equation 3, 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 heuristic loss component $\epsilon_\mathcal{H}$ and then train a cost function $\hat{\mathcal{C}}$ to optimize the cost function loss $\epsilon_{\mathcal{C}|\hat{\mathcal{H}}}$ conditioned on $\hat{\mathcal{H}}$.

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

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.

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. In fact, this approach has been explored for structured prediction in the case of greedy search (Wick et al. 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 et al. 2012), has shown that generic RL algorithms can struggle for some structured prediction problems, even with significant effort put forth by the designer. Rather, 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 problem.

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 (only known for the training data). 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, Fern, and Tadepalli 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.

Imitating Search Behavior. Given a search space over complete outputs $\mathcal{S}_o$, 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 pair-wise 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. 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. In our implementation we employed the margin-scaled variant of the online Passive-Aggressive algorithm (see Equation 47 in (Crammer et al. 2006)) as our base learner to train the heuristic function, which we have found to be quite effective yet efficient.

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, and Givan 2006; Ross and Bagnell 2010). Our experiments show, that the simple approach described above, performs extremely well on our problems.

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 that is consistent with the constraints causes the search to generate the same set of outputs for input x. The precise specification of these constraints depends on the search procedure. For space reasons, here we only describe the generation of sufficient constraints for greedy search, which we use in our experiments. Similar formulations for other rank-based search strategies is straightforward.

Greedy search simply starts at the initial state, selects the best successor state according to the heuristic, and then repeats this process for the selected state. At every search step, the best node (x, y_{best}) from the candidate set C (i.e. successors of current search state) needs to be ranked better (lower $\mathcal{H}$-value) than every other node. Therefore, we include one ranking constraint for every node $(x, y) \in C \backslash (x, y_{best})$ such that $\mathcal{H}(x, y_{best}) < \mathcal{H}(x, y)$.

Cost Function Learning

Given a learned heuristic $\mathcal{H}$, we now want to learn a cost function that correctly ranks the potential outputs generated using $\mathcal{H}$. More formally, let l_{best} be the loss of the best output among those outputs as evaluated by the true loss function L, i.e., $l_{best} = min_{y\in\mathcal{Y}_{\mathcal{H}}(x)}\ L(x, y, y^*)$. The specific goal of learning is, then, 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 $\hat{y} = \arg min_{y \in \mathcal{Y}_{\mathcal{H}}(x)}\ \mathcal{C}(x, y)$.

We formulate the cost function training problem similarly to traditional *learning to rank* problems (Agarwal and 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 $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$.

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 1) 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. We use the online Passive-Aggressive algorithm (Crammer et al. 2006) as our rank learner, exactly as configured for the heuristic learning.

Algorithm 1 Cost Function Learning via Cross Validation

```
Input: training examples $T = \{\langle x, y^* \rangle\}$, search strategy $\mathcal{A}$, time bound $\tau$, loss function $L$.
1: Divide the training set T into $k$ folds such that
   $T = \cup_{i=1}^{k} fold_i$
2: Learn $k$ different heuristics $\mathcal{H}_1, \cdots, \mathcal{H}_k$:
   $\mathcal{H}_i$ = LearnH($T_i, L, \mathcal{A}, \tau$), where $T_i = \cup_{j \neq i} fold_j$
3: Generate ranking examples for cost function training using each heuristic $\mathcal{H}_i$:
   $R_i$ = Generate-CL-Examples($fold_i, \mathcal{H}_i, \mathcal{A}, \tau, L$)
4: Train cost function on all the ranking examples:
   $\mathcal{C}$ = Rank-Learner($\cup_{i=1}^{k} R_i$)
```

4 Experiments and Results

Datasets. We evaluate our approach on the following four structured prediction problems (three benchmark sequence labeling problems and a 2D image labeling problem): **1) 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, Guestrin, and Koller 2003). We consider two different variants of this task as in (Hal Daumé III, Langford, and Marcu 2009); in `HW-Small` version, we use one fold for training and remaining 9 folds for testing, and vice-versa in `HW-Large`. **2) NETtalk Stress.** 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. **3) 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. **4) Scene labeling.** This dataset contains 700 images of outdoor scenes (Vogel and Schiele 2007). Each image is divided into patches by placing a regular grid of size 10×10 and each patch takes one of the 9 semantic labels (*sky, water, grass, trunks, foliage, field, rocks, flowers, sand*). Simple appearance features like 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.

Experimental setting. For our $\mathcal{HC}$-Search experiments, we use the Limited Discrepancy Space (LDS) exactly as described in (Doppa, Fern, and Tadepalli 2012) as our search space over structured outputs. Prior work (Doppa, Fern, and Tadepalli 2012) and our own experience with $\mathcal{HC}$-Search has shown that greedy search works quite well for most structured prediction tasks, particularly when using the LDS space. Hence, we consider only greedy search in our experiments, noting 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 15 search steps for all domains except for scene labeling, which has a much larger search space and uses $\tau = 150$. 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 in all cases.

Comparison to state-of-the-art. We compare the results of our **$\mathcal{HC}$-Search** approach with other structured prediction algorithms including **CRFs** (Lafferty, McCallum, and Pereira 2001), **SVM-Struct** (Tsochantaridis et al. 2004), **Searn** (Hal Daumé III, Langford, and Marcu 2009), **Cascades** (Weiss and Taskar 2010) and **$\mathcal{C}$-Search**, the most similar approach to ours, which uses a single-function for output space search (Doppa, Fern, and Tadepalli 2012). We also show the performance of **Recurrent**, which is a simple recurrent classifier trained exactly as in (Doppa, Fern, and Tadepalli 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, Searn and Cascades (see (Doppa, Fern, and Tadepalli 2012)). 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. 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.

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

Table 1: Error rates of different structured prediction algorithms.

ALGORITHMS	DATASETS				
	HW-Small	HW-Large	Stress	Phoneme	Scene labeling
$\mathcal{HC}$-Search	**13.35**	**3.78**	**17.44**	**16.05**	**19.71**
$\mathcal{C}$-Search	17.41	7.41	21.15	20.91	27.05
CRF	19.97	13.11	21.48	21.09	-
SVM-Struct	19.64	12.49	22.01	21.7	-
Recurrent	34.33	25.13	27.18	26.42	43.36
Searn	17.88	9.42	23.85	22.74	37.69
Cascades	30.38	12.05	22.82	30.23	-

Third-Order Features

$\mathcal{HC}$-Search	**11.24**	**3.03**	**16.61**	**14.75**	**18.25**
$\mathcal{C}$-Search	14.15	4.92	19.79	18.31	25.79
Cascades	18.82	6.24	26.52	31.02	-

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

DATASETS	HW-Small			HW-Large			Stress			Phoneme			Scene		
ERROR	ϵ	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	ϵ	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	ϵ	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	ϵ	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$	ϵ	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\|\mathcal{H}}$
$\mathcal{HC}$-Search	13.35	5.47	7.88	3.78	1.34	2.44	17.44	3.13	14.31	16.05	3.98	12.07	19.71	5.82	13.89
$\mathcal{C}$-Search	17.41	5.59	11.82	7.41	1.51	5.9	21.15	3.24	17.91	20.91	4.38	16.53	27.05	7.83	19.22
$\mathcal{LC}$-Search (Oracle $\mathcal{H}$)	10.12	0.22	9.9	3.07	0.53	2.54	14.19	0.26	13.93	12.25	0.51	11.74	16.39	0.37	16.02

Cascades[1]. 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 compare 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. In fact, $\mathcal{HC}$-Search using only second-order features is still outperforming the third-order results of the other methods across the board. Finally, we note that while Cascades is able to significantly improve performance in two of its four data sets by using third-order features, in the other two benchmarks performance degrades.

Loss Decomposition Analysis. We now examine $\mathcal{HC}$-Search and $\mathcal{C}$-Search in terms their loss decomposition (see Equation 3) 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 ϵ 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.

Oracle Heuristic Results. Noting that there is room to improve the generation loss, the above results do not indicate whether improving this loss via a better learned heuristic would lead to better results overall. 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 $\mathcal{LC}$-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 note that we have experimented with more sophisticated imitation learning algorithms (e.g., Dagger (Ross, Gordon,

[1]http://code.google.com/p/structured-cascades/

and Bagnell 2011)), but did not see significant improvements.

We also see from the oracle results that the overall error ϵ is better than that of $\mathcal{HC}$-Search. 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.

5 Conclusions 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 heuristic. This analysis suggests that learning more powerful cost functions, e.g., Regression trees (Mohan, Chen, and Weinberger 2011), with an eye towards anytime performance (Grubb and Bagnell 2012; Xu, Weinberger, and Chapelle 2012) would be productive. Our results also suggested that there is also 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).

Acknowledgements

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.

References

Agarwal, S., and Roth, D. 2005. Learnability of bipartite ranking functions. In *COLT*, 16–31.

Collins, M. 2000. Discriminative reranking for natural language parsing. In *ICML*, 175–182.

Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; and Singer, Y. 2006. Online passive-aggressive algorithms. *JMLR* 7:551–585.

Dietterich, T. G.; Hild, H.; and Bakiri, G. 1995. A comparison of ID3 and backpropagation for english text-to-speech mapping. *MLJ* 18(1):51–80.

Doppa, J. R.; Fern, A.; and Tadepalli, P. 2012. Output space search for structured prediction. In *ICML*.

Felzenszwalb, P. F., and McAllester, D. A. 2007. The generalized A* architecture. *JAIR* 29:153–190.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *JAIR* 25:75–118.

Fern, A. 2010. Speedup learning. In *Encyclopedia of Machine Learning*. 907–911.

Grubb, A., and Bagnell, D. 2012. Speedboost: Anytime prediction with uniform near-optimality. *JMLR Proceedings Track* 22:458–466.

Hal Daumé III; Langford, J.; and Marcu, D. 2009. Search-based structured prediction. *MLJ* 75(3):297–325.

Hoffgen, K.-U.; Simon, H.-U.; and Horn, K. S. V. 1995. Robust trainability of single neurons. *Journal of Computer and System Sciences* 50(1):114–125.

Jiang, J.; Teichert, A.; Daumé III, H.; and Eisner, J. 2012. Learned prioritization for trading off accuracy and speed. In *NIPS*, 1340–1348.

Khardon, R. 1999. Learning to take actions. *Machine Learning* 35(1):57–90.

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 441–448.

Mohan, A.; Chen, Z.; and Weinberger, K. Q. 2011. Web-search ranking with initialized gradient boosted regression trees. *JMLR Proceedings Track* 14:77–89.

Ross, S., and Bagnell, D. 2010. Efficient reductions for imitation learning. In *AISTATS*, 661–668.

Ross, S.; Gordon, G.; and Bagnell, D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 627–635.

Taskar, B.; Guestrin, C.; and Koller, D. 2003. Max-margin markov networks. In *NIPS*.

Tsochantaridis, I.; Hofmann, T.; Joachims, T.; and Altun, Y. 2004. Support vector machine learning for interdependent and structured output spaces. In *ICML*.

Vogel, J., and Schiele, B. 2007. Semantic modeling of natural scenes for content-based image retrieval. *IJCV* 72(2):133–157.

Weiss, D., and Taskar, B. 2010. Structured prediction cascades. In *AISTATS*, 916–923.

Weiss, D.; Sapp, B.; and Taskar, B. 2010. Sidestepping intractable inference with structured ensemble cascades. In *NIPS*, 2415–2423.

Wick, M. L.; Rohanimanesh, K.; Singh, S.; and McCallum, A. 2009. Training factor graphs with reinforcement learning for efficient map inference. In *NIPS*, 2044–2052.

Wick, M. L.; Rohanimanesh, K.; Bellare, K.; Culotta, A.; and McCallum, A. 2011. Samplerank: Training factor graphs with atomic gradients. In *ICML*, 777–784.

Xu, Y.; Fern, A.; and Yoon, S. 2009. Learning linear ranking functions for beam search with application to planning. *JMLR* 10:1571–1610.

Xu, Z.; Weinberger, K.; and Chapelle, O. 2012. The greedy miser: Learning under test-time budgets. In *ICML*.