AN ABSTRACT OF THE DISSERTATION OF

Janardhan Rao Doppa for the degree of Doctor of Philosophy in Computer Science presented on July 17, 2014.

Title: Integrating Learning and Search for Structured Prediction

Abstract approved: _____

Prasad Tadepalli Alan P. Fern

We are witnessing the rise of the data-driven science paradigm, in which massive amounts of data – much of it collected as a side-effect of ordinary human activity – can be analyzed to make sense of the data and to make useful predictions. To fully realize the promise of this paradigm, we need automated systems that can transform structured inputs to structured outputs. Examples include parsing a sentence, resolving coreferences of entity and event mentions in a piece of text, interpreting a visual scene, and translating from one language to another. Problems such as these are often referred to as structured prediction problems in the machine learning community. These prediction problems pose severe learning and inference challenges due to the huge number of possible outputs.

This thesis explores how to integrate two fundamental branches of Artificial Intelligence, namely *learning* and *search*, to solve structured prediction tasks. We study a new framework for structured prediction called $\mathcal{HC}$-Search, where we formulate the problem of structured prediction as an explicit search process in the combinatorial space of outputs. The system starts from a reasonably good initial solution and performs an heuristic search guided by a learned heuristic function $\mathcal{H}$ until a fixed number of alternative solutions has been generated or a fixed time limit is reached. It then evaluates each of these alternatives using a learned cost function $\mathcal{C}$ and returns the minimum-cost solution.

There are three key learning challenges in this framework – Search space design: how can we automatically *design* an efficient search space over structured outputs?; Heuristic learning: how can we learn a heuristic function $\mathcal{H}$ for effectively *guiding* the search?; Cost function learning: how can we learn a cost function $\mathcal{C}$ that can accurately *select* the best output among the

candidate outputs? We develop generic solutions for each of these learning challenges and an engineering methodology for applying this framework. We show that the $\mathcal{HC}$-Search framework achieves results in a wide range of structured prediction problems that significantly exceed the best previous results.

Integrating Learning and Search for Structured Prediction

by

Janardhan Rao Doppa

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 17, 2014
Commencement June 2015

Doctor of Philosophy dissertation of Janardhan Rao Doppa presented on July 17, 2014.

APPROVED:

Co-Major Professor, representing Computer Science

Co-Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Janardhan Rao Doppa, Author

ACKNOWLEDGEMENTS

I was extremely fortunate to have a lot of incredibly nice and helpful people in my life and it is very important to thank all these wonderful people. However, Prasad Tadepalli, Alan Fern, and Tom Dietterich are very important and special people to me, and they will remain so for the rest of my life. I owe everything I learned in graduate school to them. They gave me very good training and mentored me all the way through this journey.

I would like to begin by thanking my advisor Prasad Tadepalli for his help, guidance, and the freedom he gave to pursue my ideas over the years. As a young graduate student, I made a lot of mistakes (in the hindsight), partly because I didn't know the "academic research MDP." Prasad showed immense faith in me during that difficult phase. If not for his faith and patience, the research community might not have seen the "$\mathcal{HC}$-Search framework." In spite of his busy schedule, he was very approachable and patiently listened to all my premature ideas and problem formulations, and provided critical feedback on my research work. He always amazed me with his fundamental questions full of insights that greatly helped in improving the quality of my work. I also highly appreciate his support and help throughout my academic job search. He always put his students above everything, which is something I will try hard to achieve. He deeply cared about the well-being of everyone around him including myself, and along the way inspired me to become a better human being.

I would like to thank my co-advisor, the *inmitable* Alan Fern, for providing me high-quality training data at all stages of the research life-cycle, and for his professional advice. I learned most of my core research skills from Alan via imitation learning. He taught me how to break a complex problem into simpler ones to quickly get to the bottom of it; how to productively explore the space of solutions to quickly reach a near-optimal solution; and how to tell a good story based on the audience to effectively communicate my research ideas. In spite of his extremely busy schedule, he provided timely feedback on my initial ideas and formulations, and helped me in refining them. I hope his indefatigable spirit of pursuing a problem until he sees the solution as a series of logical steps has rubbed off on me. I have seen Alan's rise since he joined OSU as a junior faculty, and he greatly inspired me with his deep research work over diverse topics. In fact, I wouldn't have pursued this line of research if I haven't read his beautiful work on beam search with his student Yuehua Xu. Thank-you, Alan, for being a great role model.

I was very fortunate to have Tom Dietterich as one of my closest mentor. He is also my closest collaborator outside this work. I worked on several diverse projects and wrote several papers with him. I learned both directly and indirectly (e.g., his Fall 1996 course on Research Methods in CS, Dietterich's rules of English) from him. He also helped me in polishing my writing skills and often emphasized the importance of good writing. His positive encouragement and support made a lot of difference to me. His enthusiasm for research and optimism are highly infectious. It was a great experience working with and learning from a visionary and a leader like him. While it will be naive to think that Tom's uncanny ability to look at the bigger picture can be inculcated, I sincerely hope that I have taken a leaf off it. In spite of his jet speed schedule, he spent a lot of time to help me with my academic job search (as much as a faculty would do for his/her direct advisee), and also created a lot of professional opportunities for me. Thanks for patiently answering all my questions related to academics. I can't imagine a better mentor than him.

A big thank-you to Bella Bose who always took the time to enquire about my work, progress, and well-being in spite of being very busy wearing multiple hats at the same time. He gave me several opportunities to learn about the academic MDP by putting me on the faculty hiring committee for two consecutive years and also giving me the opportunity to teach the 507 course on *Introduction to Graduate School*. Teaching the 507 course gave me a lot of satisfaction and I hope someone will continue teaching it in the future.

I would like to thank Dan Roth for being a great mentor and inspiring me through his deep research work on learning to reason dating back to his PhD work. I highly appreciate his help with my job search, and for his encouraging emails telling me that I did a good job and will do well as a junior faculty. Thanks to Roni Khardon who greatly influenced me in a lot of different ways. Some part of this dissertation work is built on his seminal work on learning to take actions. I enjoyed all my interactions with him, and look forward to learning more from him in the future. Thanks to Jason Eisner for that caring note he sent to me after my job search. It was very touching to receive it consdering the fact that I never collaborated with him. Thanks to Sinisa Todorovic for some of the practical advice he gave me from time to time.

A special thanks to my friend and academic sibling Sriraam Natarajan. He gave me good advice, guidance, help, and provided critical feedback when needed. He has done wonderful things after graduating from OSU and inspired many graduate students like me. Thanks to Kristian Kersting for his friendship and mentoring advice. His leadership skills are something I can only aspire for. I enjoyed the time I spent with him at various conferences.

I was very fortunate to collaborate and work with several people on a lot of diverse projects. Thanks to all my collaborators for their excellent work and fantastic ideas: Charles Parker, Chris-Mills Price, Mohammad Nasresfahani, Shahed Sorower, Chao Ma, Jun Xie, Walker Orr, Prashanth Mannem, Nathan Murrow, Rui Qin, Michael Lam, Shell Hu Xu, Jun Yu, Liping Liu, Jed Irvine, Tom Dietterich, Xiaoli Fern, Lise Getoor, Sinisa Todorovic, and Weng-Keen Wong. I would also like to thank all the machine learning reading group participants over the years. I learned a lot from all of you.

Thanks to all the great researchers who influenced me and my work over the years. In particular, I would like to thank late Ben Taskar, Dan Roth, Drew Bagnell, Roni Khardon, Andrew McCallum, Lise Getoor, Jason Eisner, Hal Daumé III, John Langford, Kristian Kersting, and Subbarao Kambhampati.

Thanks to all the OSU administrative staff including Ferne Simendinger, Nicole Thompson, Colisse Franklin, Mike Sander, Todd Schecter, Renee Lyon, and Pat Sullivan, for all your help. You were my lifelines at OSU. Special thanks goes to Mike Sander, for accommodating all my requests for cluster resources and helping me in meeting all my deadlines; and Pat Sullivan, for making the GRA appointments, conference travel, and reibursements easier for all the AI students including myself.

I would like to thank all the wonderful teachers and mentors I had before coming to OSU: Pabitra Mitra, TV Prabhakar, Sumit Ganguly, Mainak Chaudhuri, Bhaskar Ramanan, M.N. Seetaramanathan, N.B. Venkateswarlu, KVSVN Raju and M. Shashi. I would especially like to thank both MNS and NBV for regularly enquiring about my progress in graduate school over the years, and for sending cheerful notes on many occasions.

I would like to thank all the friends at OSU who made my stay at OSU very enjoyable: Rajesh Inti, Ravi Tagore, Charith Abeywarna, Radha-Krishna Balla, Arunkumar Puppala, Jervis Pinto, Rob Hess, Theresa Migler, Ethan Dereszenski, Jun Yu, Shahed Sorower, Saikat Roy, Kshitiz Judah, Sriraam Natarajan, Aaron Wilson, Ronny Bjarnson, Javad Azimi, Chris Chambers, Nadia Payet, William Brendel, Shell Hu Xu, Shubhomoy Das, Rebecca Hutchinson, and Prashanth Mannem. Thanks to Padma Akkaraju for being a motherly figure to me, for all the good advice, and for showing a lot of care towards me over the years. Thanks to Shravya Tadepalli, the walking encyclopedia, for sharing her knowledge of the world enthusiastically, and to Soumya Tadepalli for the humorous conversations.

Thanks to the cricket community in Portland and Seattle for giving me an opportunity to play and enjoy the game I love the most. I thoroughly enjoyed all the games I played over the

years for various clubs including OSU, Portland, and Chak De Oregon. Special thanks goes to my good friends Raiyo Aspandiar, Narinder, and Sonu.

A big thank-you to my very close friend Ganesh Yerubandi for his support and encouragement, and for believing in me even during my bad days. His selfless attitude and helping nature is something I can only aspire for. I couldn't have asked for a better guardian than him. Thanks also goes to my good friend Raja Sandireddy for all his support and help.

I would like to thank all my "coolandhra" friends for their encouragement, support, and providing a virtual home that is very close to my heart. I wouldn't have been able to finish this long journey without their love and affection. Thanks to my good friends from IITK, Barna Saha and Arya Mazumdar, who inspired me through their great research work. Special thanks goes to Vinaya Natarajan for being a great friend to me over the years. She virtually played a motherly role for me. Words cannot express my gratitude to her. Thanks to my close friend Vijaya Saradhi for his encouragement and support from my time at IITK. He gave me very good advice on several occasions and taught me how to be enthusiastic about your ideas and work. Thanks to Suryaprakash Kotha, Venkatrao Chimata, Late Ranjit Vasireddy, Kaushik Ramajyam, Mallika Allu, and Swati Tata for all their help when I applied to graduate schools.

On the personal side, I would like to thank my parents Penta Rao Doppa and Varalaxmi Doppa for all the sacrifices they have made to provide good education to me even though they never had any formal education themselves. All my little achievements over the years are all yours. Thank-you *Amma* and *Nanna* for your love and affection. Thanks to my brother Pavan for shouldering the family responsibility in my absence, to my sister Anasuya for her love and care, and to my sweet nephews Sai and Mahi for making me laugh over our phone conversations. Last but not the least, I would like to thank my wife Prameela for the love and comfort she provided me after our marriage. She was very understanding during the last six months when I was very busy with my job search and dissertation work. I look forward to sharing the rest of my life with her.

Finally, I would like to thank all the funding agencies that supported my research work. I greatfully acknowledge the support of National Science Foundation (NSF) through grant IIS 1219258; and Defense Advanced Research Project Agency (DARPA) through Integrated Learning (IL) project; Machine Reading (MR) project; and Deep Exploration and Filtering of Text (DEFT) project.

CONTRIBUTION OF AUTHORS

- Jun Yu and Chao Ma were involved in the research presented in Chapter 5.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

This dissertation is dedicated to all my Teachers and Mentors.

Chapter 1: Introduction

Over the last two decades, significant progress has been made in building intelligent machines. We now have machines that can accurately solve simple classification problems (e.g., classifying an email as spam or not spam, recognizing a person from an image of their face). However, computers are still not very good at solving structured prediction problems (Bakir et al., 2007). In structured prediction problems, the predictor must produce a complex structured output given a complex structured input. For example, in Part-Of-Speech (POS) tagging, the structured input is a sequence of words and the structured output consists of 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 labeling of the image regions. Structured prediction tasks 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 computational biology (e.g., protein secondary structure prediction and gene prediction).

This dissertation explores how to integrate two fundamental branches of AI, *learning* and *search*, to solve structured prediction problems. Viewed as a traditional classification problem, the set of possible classes in structured prediction is exponential in the size of the output. 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. The main contribution of this dissertation is to address this challenge by studying a broad search-based framework for structured prediction that supports learning to improve both speed and accuracy.

The standard approach to structured prediction is to learn a cost function for scoring a potential output for each input. Given such a cost function and a new input, the output computation involves solving the so-called "Argmin" *inference problem*, which is to find the minimum cost output for the corresponding input. Unfortunately, exactly solving the Argmin inference problem

is often intractable (NP-Hard) except for some special cases such as chains and trees. Existing methods such as Conditional Random Fields (CRF) (Lafferty et al., 2001) and Structured Support Vector Machines (SSVM) (Tsochantaridis et al., 2004) try to learn a cost function that can score the correct output higher than all incorrect outputs, which is a very hard ranking problem in general.

In this dissertation, we study a framework for structured prediction called $\mathcal{HC}$-Search, which is based on search in the space of complete outputs. The framework involves first defining a combinatorial search space over complete structured outputs that allows for traversal of the output space. Next, given a structured input, the system starts from the initial solution and performs an heuristic search guided by a learned heuristic function $\mathcal{H}$ until a fixed number of alternative solutions have been generated (or a fixed time limit is reached). It then evaluates each of these alternatives using a learned cost function $\mathcal{C}$ and returns the minimum-cost alternative. The key insight is that, unlike existing approaches, the cost function only needs to select from the small subset of candidate outputs generated during the search. This is much easier than the standard CRF/SSVM approach, which must learn a scoring function that ranks the correct answer above all possible alternative answers.

Advantages of $\mathcal{HC}$-Search. $\mathcal{HC}$-Search has several advantages compared to the existing approaches for structured prediction (see Chapter 2 for more details):

- The framework scales gracefully with the representation complexity. In particular, the approach only needs to be able to efficiently evaluate the heuristic and cost function at specific input-output pairs, which is generally straightforward even when the corresponding Argmin problem is intractable. Thus, we are free to increase the complexity of the $\mathcal{H}$ and $\mathcal{C}$ functions without considering the impact on inference complexity.

- Since the framework is based on search over complete outputs, our inference is inherently an anytime procedure, meaning that it can be stopped at any time and return the best output discovered so far. The framework provides a way of using machine learning to trade off accuracy for inference-time efficiency as dictated by the application. For example, we can optimize accuracy given constraints on the prediction time or minimize prediction time while maintaining high accuracy.

- The training procedure is sensitive to the particular loss function (evaluation metric) of interest and makes minimal assumptions about it, requiring only that we have a black box

that can evaluate it for any potential output. It can even work with non-decomposable loss functions (e.g., F1 score).

- The overall error decomposes into heuristic error (error due to not generating the optimal solution) and cost function error (error due to not selecting the best candidate solution generated by the heuristic). These errors can be easily measured for a learned ($\mathcal{H}$,$\mathcal{C}$) pair and allow for an assessment of which function is more responsible for the overall error. This in turn can guide additional engineering of the representation of the $\mathcal{H}$ and $\mathcal{C}$ functions.

1.1 Technical Contributions

The main contribution of this dissertation is the "$\mathcal{HC}$-Search framework" that integrates learning and search in a principled manner for solving structured prediction problems. There are three key learning challenges in this framework – Search space design: how can we automatically *design* an efficient search space over structured outputs?; Heuristic learning: how can we learn a heuristic function $\mathcal{H}$ for effectively *guiding* the search?; Cost function learning: how can we learn a cost function $\mathcal{C}$ that can accurately *select* the best output among the candidate outputs? We develop generic solutions for each of these learning challenges and an engineering methodology for applying this framework. In particular, the contributions include the following.

1. **Search Space over Structured Outputs:** We adapted the basic idea of Limited Discrepancy Search (LDS) (Harvey and Ginsberg, 1995) to structured prediction by defining the *Limited Discrepancy Search Space* (Doppa et al., 2014b), a generic search space over outputs that leverages greedy classifers (Dietterich et al., 1995; Hal Daumé III et al., 2009). These classifiers build the structured output incrementally by making a sequence of interrelated decisions in a *primitive space* (e.g., labeling a sequence from left-to-right). Such classifiers are very efficient, but of course some decisions are difficult to make by a greedy classifier, and the resulting errors can propagate to downstream decisions and lead to poor global performance. The key idea behind LDS is to note that if the classifier response can be corrected at one or more of these critical errors, then a much better output will be produced. Any such change that is made to override a decision taken by the greedy classifier is called a *discrepancy*. If the accuracy of the classifier on individual decisions is high, then the number of discrepancies needed to produce a correct output will be correspond-

ingly small. 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 smaller to larger sets. With a fairly good classifier, the target outputs can be found at a small depth in the discrepancy search tree.

2. **Heuristic Learning:** 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 (which is only available during training). Motivated by this observation, the heuristic learning problem is formulated in the framework of *imitation learning* to learn a heuristic that mimics the search decisions made by the true loss function on training examples. This is a generic approach that is applicable to all ranking-based search procedures (e.g., greedy search and beam search). We provide a characterization of the ranking constraints that are *minimally sufficient* to replicate the search behavior with true loss function. The aggregate set of ranking constraints collected over all of the training examples is given to a rank learning algorithm (e.g., Perceptron or SVM-Rank) to learn the heuristic function $\mathcal{H}$.

3. **Cost Function Learning:** Given a learned heuristic $\mathcal{H}$, we want to learn a cost function that correctly ranks the candidate outputs generated by the search procedure guided by $\mathcal{H}$. This is formulated as another rank-learning problem such that the cost function $\mathcal{C}$ scores the best output generated during search higher than the other outputs.

4. **Multi-Label Search Framework:** We developed a simple framework for multi-label prediction called Multi-Label Search (MLS) based on instantiating our $\mathcal{HC}$-Search framework to multi-label learning by *explicitly* exploiting the sparsity property of multi-label problems. We empirically evaluated our MLS framework along with many existing multi-label learning algorithms on a variety of benchmarks by employing diverse task loss functions.

1.2 Outline of the Thesis

The remaining part of the dissertation is organized as follows. In Chapter 2, we discuss the related work on structured prediction and motivate our $\mathcal{HC}$-Search approach.

We describe the $\mathcal{C}$-Search framework for structured prediction based on search in the space of complete structured outputs in Chapter 3. Given a structured input, an output is produced by

running a time-bounded search procedure guided by a learned cost function, and then returning the least cost output uncovered during the search. This framework can be instantiated for a wide range of search spaces and search procedures. In this chapter, we make two main technical contributions. First, we describe a novel approach to automatically defining an effective search space over structured outputs, which is able to leverage the availability of powerful classification learning algorithms. In particular, we define the limited-discrepancy search (LDS) space and relate the quality of that space to the quality of the learned classifiers. We also define a sparse version of the search space to improve the efficiency of our overall approach. Second, we give a generic cost function learning approach that is applicable to a wide range of search procedures. The key idea is to learn a cost function that attempts to mimic the behavior of conducting searches guided by the true loss function. Our experiments on several benchmark domains show that a small amount of search in the limited discrepancy search space is often sufficient to significantly improve on state-of-the-art structured-prediction performance. We also demonstrate significant speed improvements for our approach using sparse search spaces with little or no loss in accuracy.

In Chapter 4, we introduce the $\mathcal{HC}$-Search framework for structured prediction whose principal feature is the separation of the cost function from search heuristic. 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 $\mathcal{HC}$-Search approach significantly outperforms several state-of-the-art methods including the $\mathcal{C}$-Search approach. Our investigation showed that the main source of error of existing output-space approaches including our $\mathcal{HC}$-Search approach is the inability of the cost function to correctly rank the candidate outputs produced by the output generation process.

In Chapter 5, we consider multi-label prediction problems from a search perspective. We treat multi-label learning as a special case of structured-output prediction (SP), where each input x is mapped to a binary vector y that indicates the set of labels predicted for x. The main contri-

bution of this chapter is to investigate a simple framework for multi-label prediction called Multi-Label Search (MLS) based on instantiating our $\mathcal{HC}$-Search framework to multi-label learning by *explicitly* exploiting the sparsity property of multi-label problems. We empirically evaluate the MLS framework along with many existing multi-label learning algorithms on a variety of benchmarks by employing diverse task loss functions. Our results demonstrate that the performance of existing algorithms tends to be very similar in most cases, and that the MLS approach is comparable and often better than all the other algorithms across different loss functions.

Finally, we conclude the dissertation with the lessons learned and discuss important future directions in Chapter 6.

Chapter 2: Related Work

In this chapter, we review the related work on structured prediction and compare our $\mathcal{HC}$-Search approach with existing methods.

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 $\mathcal{D}$.

2.1 Cost Function Learning

A typical 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 then involves solving the so-called Argmin problem:

$$\hat{\mathbf{y}} = \arg\min_{\mathbf{y}} \mathcal{C}(\mathbf{x}, \mathbf{y}).$$

For example, approaches including Conditional Random Fields (CRFs) (Lafferty et al., 2001), Structured Perceptron (Collins, 2002), Max-Margin Markov Networks (Taskar et al., 2003) and Structured SVMs (Tsochantaridis et al., 2004) fall in this category. These methods represent the cost function as a linear model over template features of both $\mathbf{x}$ and $\mathbf{y}$, and learn the parameters of the cost function by making repeated calls to an Argmin inference routine to optimize various objective functions, which differ among learning algorithms (Lafferty et al., 2001; Taskar et al., 2003; Tsochantaridis et al., 2004; McAllester et al., 2010).

Unfortunately exactly solving the Argmin problem is often intractable and efficient solutions exist only in limited cases such as when the dependency structure among features forms a tree. In such cases, one might 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 (Murphy et al., 1999; Andrieu et al., 1999).

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.

Inference-free Training Methods. There are also approximate cost function learning approaches that do not employ any inference routine during training. For example, piece-wise training (Sutton and McCallum, 2009), Decomposed Learning (Samdani and Roth, 2012) and its special case pseudo-max training (Sontag et al., 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 et al., 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 and tau Yih, 2005).

More recent work has attempted to integrate (approximate) inference and cost function learning in a principled manner (Meshi et al., 2010; Stoyanov et al., 2011; Hazan and 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 et al., 2009; Qian et al., 2009). However, these approaches are not applicable for the graphical models where the sparsity assumption does not hold.

2.2 Cascade Training

An alternative approach to addressing inference complexity is cascade training (Felzenszwalb and McAllester, 2007; Weiss and Taskar, 2010; Weiss et al., 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.

2.3 Control Knowledge Learning

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. These approaches then attempt 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 $\mathbf{y}$. The approach of learning an iteratively applied classifier to avoid Argmin inference can be viewed as learning control knowledge (here a classifier) that guides a greedy search in an enormous search space. Simple training methods (e.g. Dietterich et al. (1995)) have shown good success and there are some positive theoretical guarantees (Syed and Schapire, 2010; Ross and Bagnell, 2010). However, recurrent classifiers can be prone to error propagation (Kääriäinen, 2006; Ross and Bagnell, 2010). Recent work, e.g. SEARN (Hal Daumé III et al., 2009), SMiLe (Ross and Bagnell, 2010), and DAGGER (Ross et al., 2011), attempts to address this issue using more sophisticated training techniques and has 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 (Hal Daumé III and Marcu, 2005; Daumé III, 2006; Xu et al., 2009; Huang et al., 2012; Yu et al., 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., 2009).

2.4 Output Space Search

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 score the candidate outputs. Serving these dual roles often means that the cost function needs to make unclear tradeoffs, which increases 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.

While SampleRank (Wick et al., 2011) shares with our work the idea of explicit search in the output space, there are some significant differences. The SampleRank framework is mainly focused on Monte-Carlo search, and the underlying flipbit search space, whereas our approach can be instantiated for a wide range of search spaces (e.g., LDS space that leverages powerful recurrent classifiers) and rank-based search algorithms (e.g., greedy search, beam search and best-first search). We believe that this flexibility is important since it is well-understood in the search literature that the best search space formulation and the most appropriate search algorithm change from problem to problem. In addition, the SampleRank framework is highly dependent on a hand-designed "proposal distribution" for guiding the search or effectively defining the search space. In contrast, we describe a generic approach for constructing search spaces that is shown to be effective across a variety of domains.

2.5 Re-Ranking Algorithms

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 et al., 2012), M-best modes (Park and Ramanan, 2011; Chen et al., 2013) and Determinantal Point Processes (Kulesza and Taskar, 2012).

2.6 Learning to Improve Combinatorial Optimization

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 and Moore, 2000) learns an evaluation function over the states to improve the performance of search, where the value of a state corresponds to the performance of a local search algorithm starting from that state. Zhang and 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 these 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.

Chapter 3: Limited Discrepancy Search Space for Structured Prediction

In this chapter, we study a new search-based approach to structured prediction based on search in the space of complete outputs. The approach involves first defining a combinatorial search space over complete structured outputs that allows for traversal of the output space. Next, given a structured input, a state-based search strategy (e.g., best-first or greedy search), guided by a learned cost function, is used to explore the space of outputs for a specified time bound. The least cost output uncovered by the search is then returned as the prediction. This approach is motivated by our observation that for a variety of structured prediction problems, if we use the true loss function of the structured prediction problem to guide the search (even non-decomposable losses), then high-quality outputs are found very quickly. This suggests that similar performance might be achieved if we could learn an appropriate cost function to guide search in place of the true loss function.

A potential advantage of our search-based approach, compared to most structured-prediction approaches (see Section Chapter 2), is that it scales gracefully with the complexity of the cost function dependency structure. In particular, 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, we are free to increase the complexity of the cost function without considering its impact on the inference complexity. Another potential benefit of our approach is that since the search is over complete outputs, our inference is inherently an anytime procedure, meaning that it can be stopped at any time and return the best output discovered so far. This has the flexibility of allowing for the use of more or less inference time for computing outputs as dictated by the application, with the idea that more inference time may sometimes allow for higher quality outputs.

The effectiveness of our approach for a particular problem depends critically on: 1) The identification of an effective combination of search space and search strategy over structured outputs, and 2) Our ability to learn a cost function for effectively guiding the search for high quality outputs. The main contribution of our work is to provide generic solutions to these two issues and to demonstrate their empirical effectiveness.

First, we describe the limited-discrepancy search space, as a generic search space over com-

plete outputs that can be customized to a particular problem by leveraging the power of (non-structured) classification learning algorithms. We show that the quality of this search space is directly related to the error of the learned classifiers and can be quite favorable compared to more naive search space definitions. We also define a sparse version of the search space to improve the efficiency of our approach. The sparse search space tries to reduce the branching factor while not hurting the quality of the search space too much.

Our second contribution is to describe a generic cost function learning algorithm that can be instantiated for a wide class of "ranking-based search strategies." The key idea is to learn a cost function that allows for imitating the search behavior of the algorithm when guided by the true loss function. We give a PAC bound for the approach in the realizable setting, showing a polynomial sample complexity for doing approximately as well as when guiding search with the true loss function.

Finally, we provide experimental results for our approach on a number of benchmark problems and show that even when using a relatively small amount of search, the performance is comparable or better than the state-of-the-art in structured prediction. We also demonstrate significant speed improvements of our approach when used with sparse search spaces.

The remainder of the chapter is organized as follows. In Section 3.1, we introduce our problem setup and give a high-level overview of our framework. In Section 3.2, we define two search spaces over complete outputs in terms of a recurrent classifier, relate their quality to the accuracy of the classifier, and then, define sparse search spaces to improve the efficiency of our approach. We describe our cost function learning approach in Section 3.3. Section 3.4 presents our experimental results and finally Section 3.5 provides a summary of what we learned from this work.

3.1 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 $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, drawn from an unknown target distribution, where y_i is the true output for input x_i. The goal is to return a function/predictor from structured inputs to outputs whose predicted outputs have low expected loss with respect to the target distribution. Since our algorithms will be learning cost functions

over input-output pairs 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. Intuitively these features should provide some measure of compatibility between (parts of) the structured input and output.

We consider a framework for structured prediction based on state-based search in the space of complete structured outputs. The states of the search space are pairs of inputs and outputs (x, y), representing the possibility of predicting y as the output for x. A search space over those states is specified by two functions: 1) An *initial state function* I such that $I(x)$ returns an initial search state for any input x, and 2) A *successor function* S such that for any search state (x, y), $S((x, y))$ returns a set of successor states $\{(x, y_1), \ldots, (x, y_k)\}$, noting that each successor must involve the same input x as the parent. Section 3.2 will describe our approach for automatically defining and learning search spaces.

In order to predict outputs, our framework requires two elements in addition to the search space. First, we require a cost function $\mathcal{C}$ that returns a numeric cost for any input-output pair (i.e., search state). Second, we require a search procedure $\mathcal{A}$ (e.g., greedy search or beam search) for traversing search spaces, possibly guided by the cost function. Given these elements, an input x, and a prediction time bound τ we compute an output by executing the search procedure $\mathcal{A}$ starting in the initial state $I(x)$ and guided by the cost function until the time bound is exceeded. We then return the output $\hat{y}$ corresponding to the least cost state that was uncovered during the search as the prediction for x. Figure 3.1 gives a high-level overview of our search-based framework for structured prediction.

The effectiveness of our search-based framework depends critically on the quality of the search space and the cost function. Ideally we would like the search space to be organized such that high quality outputs are as close as possible to the initial state, which allows the search procedure to uncover those outputs more easily. In addition, it is critical that the cost function is able to correctly score the generated outputs according to their true losses and also to provide effective guidance to the chosen search procedure. A key contribution of this work is to propose supervised learning mechanisms for producing both high-quality search spaces and cost functions, which are described in the next two sections respectively.

3.2 Search Spaces Over Complete Outputs

In this section we describe two search spaces over structured outputs: 1) The Flipbit space, a simple but sometimes effective baseline, and 2) The limited-discrepancy search (LDS) space,

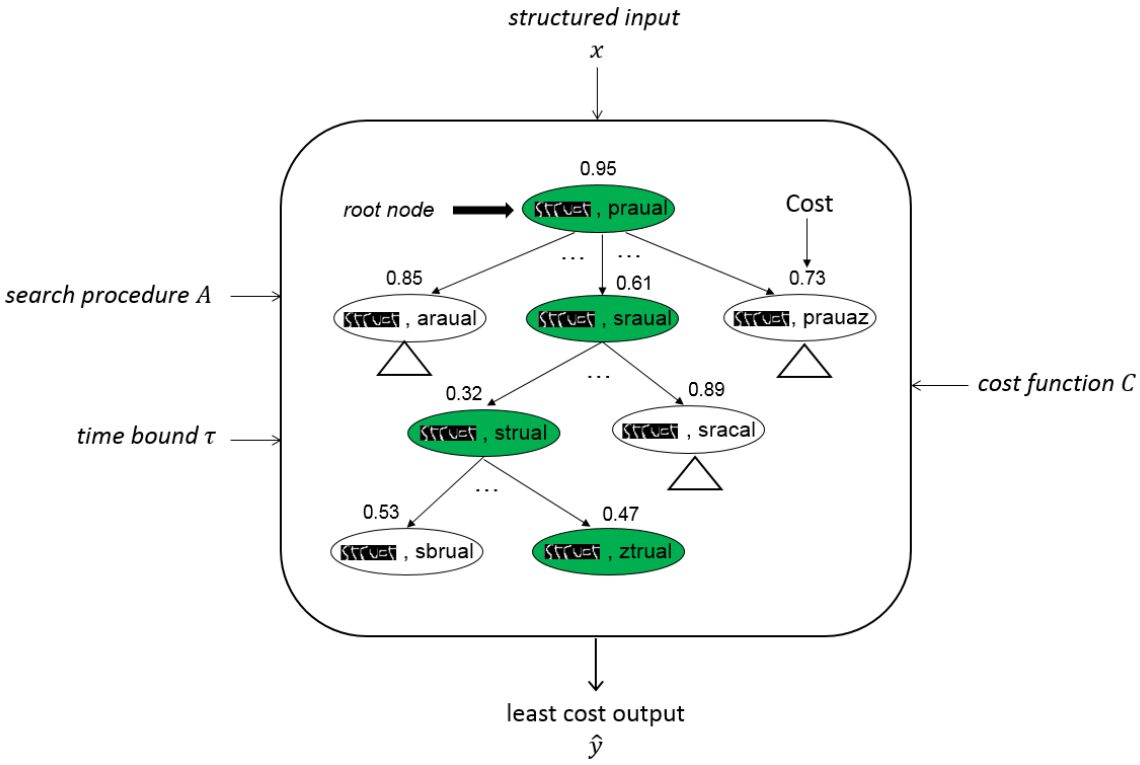

Figure 3.1: A high level overview of our output space search framework. Given a structured input x, we first instantiate a search space over complete outputs. Each search node in this space consists of a complete input-output pair. Next, we run a search procedure $\mathcal{A}$ (e.g., greedy search) guided by the cost function C for a time bound τ. The highlighted nodes correspond to the search trajectory traversed by the search procedure, in this case greedy search. We return the least cost output $\hat{y}$ that is uncovered during the search as the prediction for x.

which is intended to improve on the baseline. The key trainable element of each search space is a recurrent classifier, which once trained will fully define each space. Thus, we start this section by describing recurrent classifiers and how they are learned. Next we describe how the learned recurrent classifier is used to define each of the search spaces by defining the initial state and the successor function.

3.2.1 Recurrent Classifiers

A recurrent classifier h constructs structured outputs based on a series of discrete decisions. This is formalized for a given structured-prediction problem by defining an appropriate *primitive search space* over the possible sequences of decisions. It is important to keep in mind the distinctions between primitive search spaces, which are used by recurrent classifiers, and the search spaces over complete outputs (e.g., flipbit and LDS) upon which our overall framework is built. A primitive search space is a 5-tuple $\langle I, A, s, f, T \rangle$, where I is a function that maps an input x to an initial search node, A is a finite set of actions (or operators), s is the successor function that maps any search node and action to a successor search node, f is a feature function from search nodes to real-valued feature vectors, and T is the terminal state predicate that maps search nodes to $\{1, 0\}$ indicating whether the node is a terminal or not. Each terminal node in the search space corresponds to a *complete structured output*, while non-terminal nodes correspond to *partial structured outputs*. Thus, the decision process for constructing an output corresponds to selecting a sequence of actions leading from the initial node to a terminal. A recurrent classifier is a function that maps nodes of the primitive search space to actions, where typically the mapping is defined in terms of a feature function $f(n)$ that returns a feature vector for any search node. Thus, given a recurrent classifier, we can produce an output for x by starting at the initial node of the primitive space and following its decisions until reaching a terminal state.

As an example, for sequence labeling problems, the initial state for a given input sequence x is a node containing x with no labeled elements. The actions correspond to the selection of individual labels, and the successor function adds the selected label in the next position. Terminal nodes correspond to fully labeled sequences and the feature function computes a feature vector based on the input and previously assigned labels. Figure 3.2 provides an illustration of the primitive search space for a simple handwriting recognition problem. Each search state is a pair (x, y') where x is the structured input (binary image of the handwritten word) and y' is a partial labeling of the word. The arcs in this space correspond to search steps that label the characters in

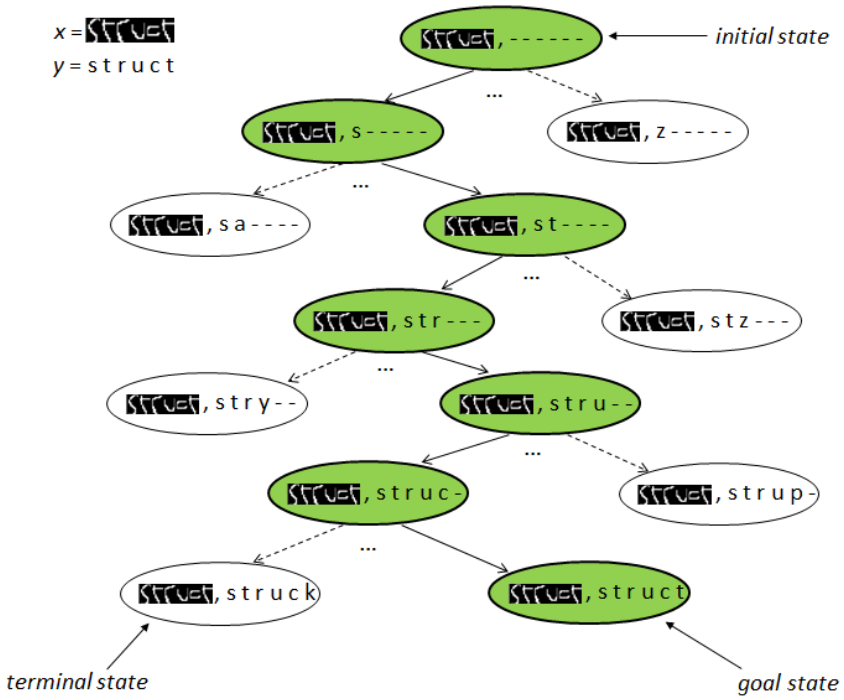

Figure 3.2: An example primitive search space for the handwriting recognition problem. Arcs represent labeling actions. Solid arcs correspond to the labeling actions taken by the recurrent classifier (optimal classifier in this case).

the input image in a left-to-right order by extending y' in all possible ways by one element. The terminal states or leaves of this space correspond to complete labelings of input x. The terminal state corresponding to the correct output y is labeled as *goal state*. Highlighted nodes correspond to the trajectory of the optimal recurrent classifier (i.e., a classifier that chooses correct action at every state leading to the goal state).

The most basic approach to learning a recurrent classifier is via *exact imitation* of the trajectory followed by the optimal classifier. For this, we assume that for any training input-output pair (x, y) we can efficiently find an action sequence, or *solution path*, for producing y from x. For example, the sequence of highlighted states in Figure 3.2 correspond to such a solution path. The exact imitation approach learns a classifier by creating a classification training example for each node n on the solution path of a structured example with feature vector $f(n)$ and label equal

to the action followed by the path at n. Our experiments will use recurrent classifiers trained via exact imitation (see Algorithm 1), but more sophisticated methods such as SEARN (Hal Daumé III et al., 2009) or DAGGER (Ross et al., 2011) could also be used.

Algorithm 1 Recurrent Classifier Learning via Exact Imitation

Input: $\mathcal{D}$ = Training examples

Output: h, the recurrent classifier

```
Initialize the set of classification examples L = ∅
for each training example (x, y = y_1 y_2 ··· y_T) ∈ D do
    for each search step t = 1 to T do
        Compute features f_n for search node n = (x, y_1 ··· y_{t-1})
        Add classification example (f_n, y_t) to L
    end for
end for
h = Classifier-Learner(L) // learn classifier from all the classification examples
return learned classifier h
```

3.2.2 Flipbit Search Space

The *Flipbit search space* is a simple baseline space over complete outputs that uses a given recurrent classifier h for bootstrapping the search. Each search state is represented by a sequence of actions in the primitive space ending in a terminal node representing a complete output. The initial search state corresponds to the actions selected by the classifier, so that $I(x)$ is equal to $(x, h(x))$, where $h(x)$ is the complete output generated by the recurrent classifier. The search steps generated by the successor function can change the value of one action at any sequence position of the parent state. In a sequence labeling problem, this corresponds to initializing to the recurrent classifier output and then searching over flips of individual labels. The flipbit space is often used by local search techniques (without the classifier initialization) and is similar to the search space underlying Gibbs Sampling.

Figure 3.3 provides an illustration of the flipbit search space via the same handwriting recognition example that was used earlier. 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

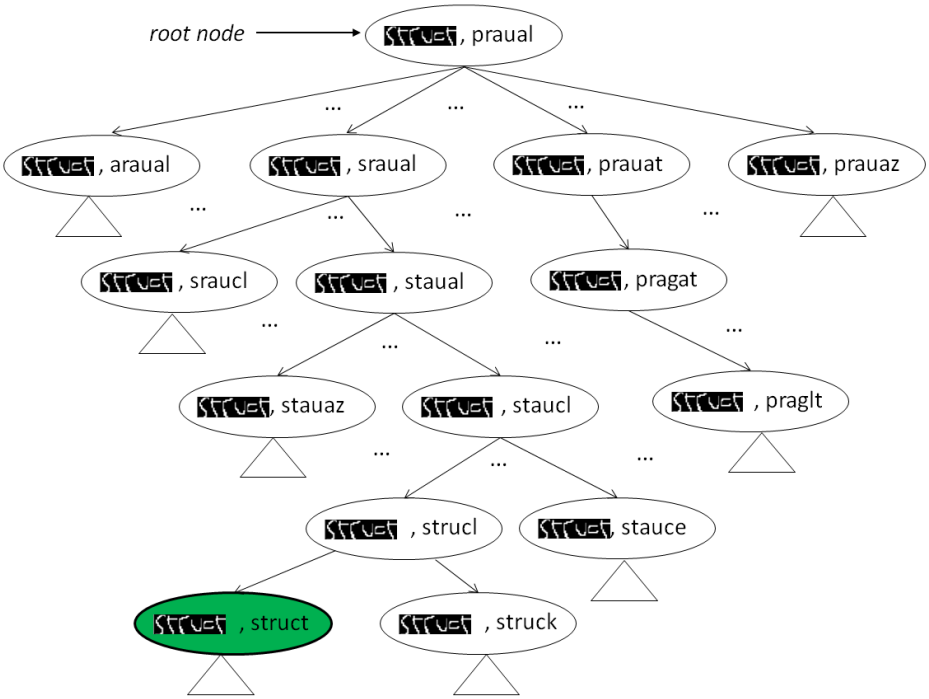

Figure 3.3: An example Flipbit search space for the handwriting recognition problem

to the number of errors in the output produced by the recurrent classifier.

3.2.3 Limited-Discrepancy Search Space (LDS)

Notice that the Flipbit space only uses the recurrent classifier when initializing the search. The motivation behind the Limited Discrepancy Search (LDS) space is to more aggressively exploit the recurrent classifier in order to improve the search space quality. LDS was originally introduced in the context of problem solving using heuristic search (Harvey and Ginsberg, 1995). To put LDS in context, we will describe it in terms of using a classifier for structured prediction given a primitive search space. If the learned classifier is accurate, then the number of incorrect action selections will be relatively small. However, even a small number of errors can propagate and cause poor outputs. The key idea behind LDS is to realize that if the classifier response was corrected at the small number of critical errors, then a much better output would be produced. LDS conducts a (shallow) search in the space of possible corrections in the hope of finding a solution better than the original.

More formally, given a recurrent classifier h and its selected action sequence of length T, a discrepancy is a pair (i, a) where $i \in \{1, \ldots, T\}$ is the index of a decision step and $a \in A$ is an action, which generally is different from the choice of the classifier at step i. For any set of discrepancies D we let $h[D]$ be a new classifier that selects actions identically to h, except that it returns action a at decision step i if $(i, a) \in D$. Thus, the discrepancies in D can be viewed as overriding the preferred choice of h at particular decision steps, possibly correcting errors, or introducing new errors. For a structured input x, we will let $h[D](x)$ denote the output returned by $h[D]$ for the search space conditioned on x. At one extreme, when D is empty, $h[D](x)$ simply corresponds to the output produced by the recurrent classifier. At the other extreme, when D specifies an action at each step, $h[D](x)$ is not influenced by h at all and is completely specified by the discrepancy set. In practice, when h is reasonably accurate, we will be primarily interested in small discrepancy sets relative to the size of the decision sequence. In particular, if the error rate of the classifier on individual decisions is small, then the number of corrections needed to produce a correct output will be correspondingly small. 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.

Consider the handwriting recognition example in Figure 3.4. The actual output produced by the classifier for the input image is `praual`, that is, output produced by introducing zero

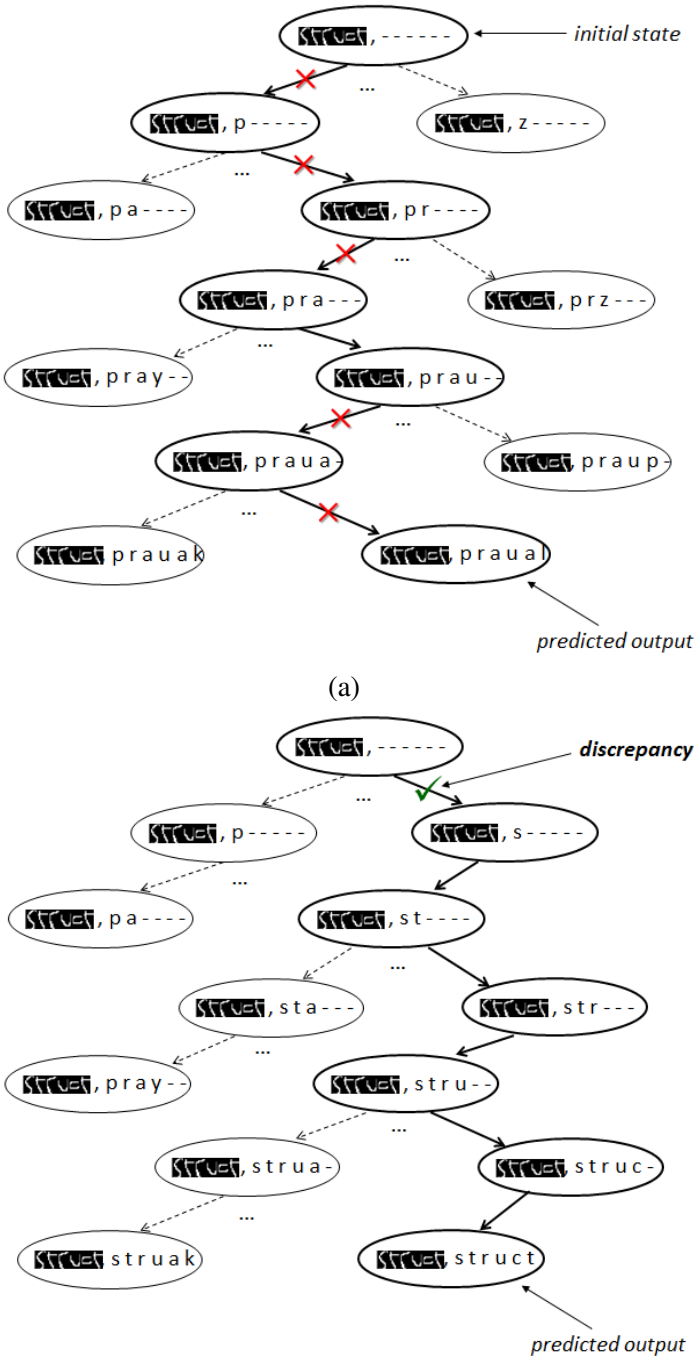

Figure 3.4: Illustration of Limited Discrepancy Search: (a) Trajectory of the recurrent classifier with no discrepancies. Arcs with 'X' mark indicate incorrect actions chosen by the classifier. (b) Trajectory of the recurrent classifier with a correction (discrepancy) at the first error. A single correction allows the classifier to correct all the remaining errors.

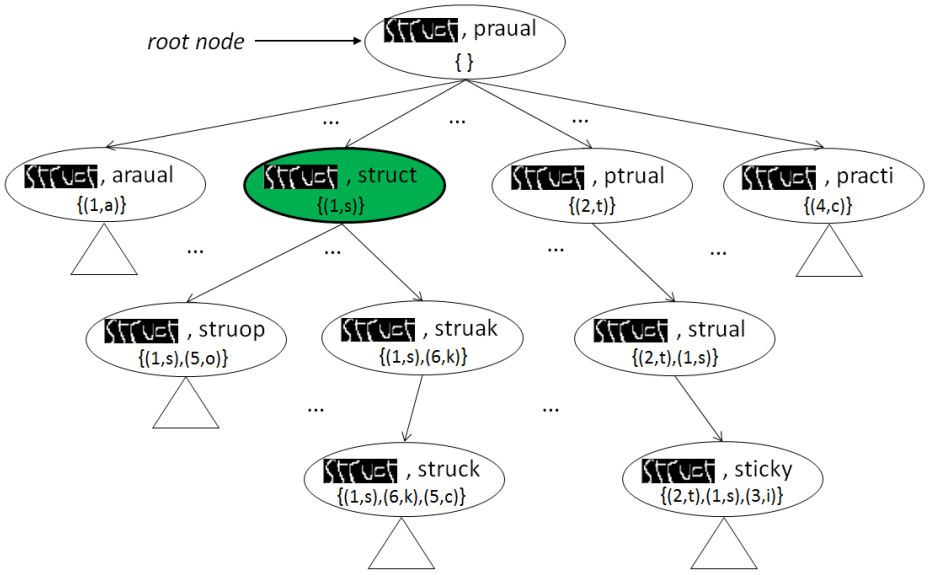

Figure 3.5: An example Limited Discrepancy Search (LDS) space for the handwriting recognition problem

discrepancies (see Figure 3.4(a)). If we introduce one discrepancy at the first position $(1, s)$ and run the classifier for the remaining labeling, we get `struct` which corrects all the remaining mistakes (see Figure 3.4(b)). By introducing this single correction, the classifier automatically corrected a number of other previous mistakes, which had been introduced due to propagation of the first error. Thus only one discrepancy was required to produce the correct output even though the original output contained many more errors.

Given a recurrent classifier h, we define the corresponding limited-discrepancy search space over complete outputs S_h as follows. Each search state in the space is represented as (x, D) where x is a structured input 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 3.5 illustrates[1] 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, that is, $h[D](x)$. The root node has an empty discrepancy set. Nodes at level one contain discrepancy sets of size one and nodes at level two contain discrepancy sets of size two, and so on. The highlighted state corresponds to the smallest depth state containing the true output.

3.2.4 Search Space Quality

Recall that in our experiments we train recurrent classifiers via exact imitation, which is an extremely simple approach compared to more elaborate methods such as SEARN. We now show the desirable property that the "exact imitation accuracy" optimized by that approach is directly related to the "quality" of the LDS search space, where quality relates the expected amount of search needed to uncover the target output. More formally, given an input-output pair (x, y) we define the *LDS target depth* for an example (x, y) and recurrent classifier h to be the minimum depth of a state in the LDS space corresponding to y. Given a distribution over input-output pairs we let $d(h)$ denote the expected LDS target depth of a classifier h. Intuitively, the depth of a state in a search space is highly related to the amount of search time required to uncover the node (exponentially related for exhaustive search, and at least linearly related for more greedy search). Thus, we will use $d(h)$ as a measure of the quality of the LDS space. We now relate $d(h)$ to the classifier error rate.

For simplicity, assume that all decision sequences for the structured-prediction problem have a fixed length T and consider an input-output pair (x, y), which has a corresponding sequence of actions that generates y. Given a classifier h, we define its *exact imitation error* on (x, y) to be e/T where e is the number of mistakes h makes at nodes along the action sequence of (x, y) (i.e., how often does it disagree with the optimal classifier along the path of the optimal classifier). Further, given a distribution over input-output pairs, we let $\epsilon_{ei}(h)$ denote the expected exact imitation error with respect to examples drawn from the distribution. Note that the exact imitation training approach aims to learn a classifier that minimizes $\epsilon_{ei}(h)$. Also, let $\epsilon_r(h)$ denote

[1] 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.

the *expected recurrent error* of h, which is the expectation over randomly drawn (x, y) of the Hamming distance between the action sequence produced by h when applied to x and the true action sequence for (x, y). The error $\epsilon_r(h)$ is the actual measure of performance of h when applied to structured prediction. Recall that due to error propagation it is possible that $\epsilon_r(h)$ can be much worse than $\epsilon_{ei}(h)$, by as much as a factor of T (e.g., see Ross et al., 2011). The following proposition shows that $d(h)$ is related to $\epsilon_{ei}(h)$ rather than the potentially much larger $\epsilon_r(h)$.

Proposition 1. *For any classifier h and distribution over structured input-outputs, $d(h) = T\epsilon_{ei}(h)$.*

Proof. For any example (x, y) the depth of y in S_h is equal to the number of imitation errors made by h on (x, y). To see this, simply create a discrepancy set D that contains a discrepancy at the position of each imitation error that corrects the error. This set is at a depth equal to the number of imitation errors and the classifier $h[D]$ will exactly produce the action sequence that corresponds to y. The result follows by noting that the expected number of imitation errors is equal to $T\epsilon_{ei}(h)$. □

It is illustrative to compare this result with the Flipbit space. Let $d'(h)$ be the expected target depth in the Flipbit space of a randomly drawn (x, y). It is easy to see that $d'(h) = T\epsilon_r(h)$ since each search step can only correct a single error and the expected number of errors of the action sequence at the initial node is $T\epsilon_r(h)$. Since in practice and in theory $\epsilon_r(h)$ can be substantially larger than $\epsilon_{ei}(h)$ (by as much as a factor of T), this shows that the LDS space will often be superior to the baseline Flipbit space in terms of the expected target depth. For example, the target depth of the example LDS space in Figure 3.5 is one and is much smaller than the target depth of the example flipbit space in Figure **??**, which is equal to five. Since this depth relates to the difficulty of search and cost-function learning, we can expect the LDS space to be advantageous when $\epsilon_r(h)$ is larger than $\epsilon_{ei}(h)$. In our experiments, we will see that this is indeed the case.

3.2.5 Sparse Search Spaces

In this section, we first discuss some of the scalability issues that arise in our framework due to the use of LDS and Flipbit spaces as defined above. Next, we describe how to define sparse versions of these search spaces to improve the efficiency of our approach.

In our search-based framework, the most computationally demanding part is the generation of candidate states during the search process. Given a parent state, the number of successor states for both the LDS and flipbit spaces is equal to $T \cdot (L-1)$ where T is the size of the structured output and L is the number of primitive action choices (the number of labels for sequence labeling problems). When T and/or L is large the time required for this expansion and computing the feature vector for each successor can be non-trivial. While we cannot control T, since that is dictated by the size of the input, we can consider reducing the effective size of L via pruning. Intuitively, for many sequence positions of a structured output, there will often be labels that can be easily determined to be bad by looking at the confidence of the recurrent classifier. By explicitly pruning those labels from consideration we can arrive at a sparse successor set that requires significantly less computation to generate.

More formally, our approach for defining a sparse successor function assumes that the recurrent classifier used to define the LDS and flipbit spaces produces confidence scores, rather than just simple classifications. This allows us to provide a ranking of the potential actions, or labels, at each position of the structured output. Using this ranking information it is straightforward to define sparse versions of the LDS and flipbit successor function by only considering successors corresponding to the top k labels at each sequence position. For the flipbit space this means that successors correspond to any way of changing the choice of the recurrent classifier at a sequence position to one of the top k labels at that position. For the LDS space, this means that we only consider introducing discrepancies at position i involving the top k labels at position i. In this way, the number of successors of a state in either space will be $T \cdot k$ rather than $T \cdot (L-1)$. Therefore, these sparse search spaces will lead to significant speed improvements for problems with large L (e.g., POS tagging, Handwriting recognition, and Phoneme prediction).

The potential disadvantage of using a small value of k is that accuracy could be hurt if good solution paths are pruned away due to inaccuracy of the classifier's confidence estimates. Thus, k provides a way to trade-off prediction speed versus accuracy. As we will show later in the experiments, we are generally able to find values of k that lead to significant speedups with little loss in accuracy.

3.3 Cost Function Learning

In this section, we describe a generic framework for cost function learning that is applicable for a wide range of search spaces and search strategies. This approach is motivated by our

empirical observation that for a variety of structured prediction problems, we can uncover high quality outputs if we guide the output-space search using the true loss function as an oracle cost function to guide the search (close to zero error with both LDS and Flipbit spaces). Since the loss function depends on correct target output y^*, which is unknown at test time, we aim to learn a cost function that mimics this oracle search behavior on the training data without requiring knowledge of y^*. With an appropriate choice of hypothesis space of cost functions, good performance on the training data translates to good performance on the test data.

3.3.1 Cost Function Learning via Imitation Learning

Recall that in our output space search framework, the role of the cost function $\mathcal{C}$ is to evaluate the complete outputs that are uncovered by the search procedure. These evaluations may be used internally by the search procedure as a type of heuristic guidance and also used when the time bound is reached to return the least cost output that has been uncovered as the prediction. Based on our empirical observations, it is very often the case that the true loss function serves these roles very effectively, which might suggest a goal of learning a cost function that is approximately equal to the true loss function L over all possible outputs. However, this objective will often be impractical and fortunately is unnecessary. In particular, the learned cost function need not approximate the true loss function uniformly over the output space, but only needs to make the decisions that are sufficient for leading the time-bounded search to behave as if it were using the true loss function. Often this allows for a much less constrained learning problem, for example, $\mathcal{C}$ may only need to preserve the rankings among certain outputs, rather than exactly matching the values of L. The key idea behind our cost learning approach is to learn such a sufficient $\mathcal{C}$. The main assumptions made by this approach are: 1) the true loss function can provide effective guidance to the search procedure by making a series of ranking decisions, and 2) we can learn to imitate those ranking decisions sufficiently well.

Our goal now is to learn a cost function that causes the search to behave as if it were using loss function L for guiding the search and selecting the final output. We propose to formulate and solve this problem in the framework of imitation learning. In traditional imitation learning, the goal of the learner is to learn to imitate the behavior of an expert performing a sequential-decision making task in a way that generalizes to similar tasks or situations. Typically this is done by collecting a set of trajectories of the expert's behavior on a set of training tasks. Then supervised learning is used to find a policy that can replicate the decisions made on those

trajectories. Often the supervised learning problem corresponds to learning a classifier or policy from states to actions and off-the-shelf tools can be used.

In our cost function learning problem, the expert corresponds to the search procedure $\mathcal{A}$ using the loss function L for a search time bound τ_{max}. The behavior that we would like to imitate is the internal behavior of this search procedure, which consists of all decisions made during the search including the final decision of which output to return. Thus, the goal of cost function learning is to learn the weights of $\mathcal{C}$ so that this behavior is replicated when it is used by the search procedure in place of L. We propose to achieve this goal by directly monitoring the expert search process on all of the structured training examples and generating the set of constraints on L that were responsible for the observed decisions. Then we attempt to learn a $\mathcal{C}$ that satisfies the constraints using an optimization procedure.

Algorithm 2 Cost 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: $\mathcal{C}$, the cost 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*
  y_{best} =**OutputOf**(s_0) *// best loss output so far*
  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**$(\mathcal{A}, L, M_{t-1} \cup C_t \setminus N_t)$
    Update the best loss output y_{best} *// track the best output*
    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
$\mathcal{C}$ =**Rank-Learner**$(\mathcal{R})$ *// learn cost function from all the ranking examples*
return learned cost function $\mathcal{C}$

Algorithm 2 describes our generic approach for cost function learning via exact imitation of searches conducted 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) $\mathcal{D} = \{(x, y^*)\}$, set of training examples for a structured prediction problem (e.g., handwriting recognition); 2) $\mathcal{S}_o = (I, S)$, definition of 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); and 5) τ_{max}, the search time bound (e.g., number of search steps).

First, it runs the search procedure $\mathcal{A}$ with the given loss function L, in the search space $\mathcal{S}_o$ instantiated for every training example (x, y^*), upto the maximum time bound τ_{max} (steps 3-10), and generates a set of pair-wise ranking examples that need to be satisfied to be able to imitate the search behavior with loss function (step 11). Second, the aggregate set of ranking examples collected over all the training examples is then given to a rank-learning algorithm (e.g., Perceptron or SVM-Rank) to learn the weights of the cost function (step 15).

The algorithmic description assumes a best-first search procedure with some level of pruning (e.g., greedy search and best-first beam search). These search procedures typically involve three key steps: 1) Selection, 2) Expansion and 3) Pruning. During selection, the search procedure selects one or more[2] open nodes from its internal memory for expansion (step 7), and expands all the selected nodes to generate the candidate set (step 8). It retains only a subset of all the open nodes after expansion in its internal memory and prunes away all the remaining ones (step 9). For example, greedy search maintains only the best node and best-first beam search with beam width b retains the best b nodes.

The most important step in our cost function learning algorithm is the generation of ranking examples to imitate the search procedure (step 11). In what follows, we first formalize ranking-based search that allows us to specify what these pairwise ranking examples are and then, give a generic description of "sufficient" pair-wise decisions to imitate the search, and illustrate them for greedy search and best-first beam search through a simple example.

3.3.2 Ranking-based Search

We now precisely define the notion of "guiding the search" with a loss function. If the loss function can be invoked arbitrarily by the search procedure, for example, evaluating and comparing

[2]Breadth-first beam search expands all nodes in the beam.

arbitrary expressions involving the cost, then matching its performance would require the cost function to approximate it arbitrarily closely, which is quite demanding in most cases. Hence, we restrict ourselves to ranking-based search defined as follows.

Let $\mathcal{P}$ be an anytime search procedure that takes an input $x \in \mathcal{X}$, calls a cost function $\mathcal{C}$ over the pairs from $\mathcal{X} \times \mathcal{Y}$ some number of times and outputs a structured output $y_{best} \in \mathcal{Y}$. We say that $\mathcal{P}$ is a ranking-based search procedure if the results of calls to $\mathcal{C}$ are only used to compare the relative values for different pairs (x, y) and (x, y') with a fixed tie breaker. Each such comparison with tie-breaking is called a ranking decision and is characterized by the tuple (x, y, y', d), where d is a binary decision that indicates y is a better output than y' for input x. When requested, it returns the best output y_{best} encountered thus far as evaluated by the cost function.

Note that the above constraints prohibit the search procedure from being sensitive to the absolute values of the cost function for particular search states (x, y) pairs, and only consider their relative values. Many typical search strategies such as greedy search, best-first search, and beam search satisfy this property.

A *run* of a ranking-based search is a sequence $x, m_1, o_1, \ldots, m_n, o_n, y$, where x is the input to the predictor, y is the output, and m_i is the internal memory state of the predictor just before the i^{th} call to the ranking function. o_i is the i^{th} ranking decision (x, y_i, y'_i, d_i). Given a hypothesis space $\mathcal{H}$ of cost functions, the cost function learning works as follows. It runs the search procedure $\mathcal{P}$ on each training example (x, y^*) for a maximum search time bound of τ_{max} substituting the loss function $L(x, y, y^*)$ for the cost function $\mathcal{C}(x, y)$. For each run, it records the set of all ranking decisions (x, y_i, y'_i, d_i). The set of all ranking decisions from all the runs is given as input to a binary classifier, which finds a cost function $C \in \mathcal{H}$, consistent with the set of all such ranking decisions.

The ranking-based search can be viewed as a Markov Decision Process (MDP), where the internal states of the search procedure correspond to the states of the MDP, and the ranking decision is an action. The following theorem can be proved by adapting the proof of Fern et al. (2006) with minor changes, for example, no discounting, and two actions, and applies to stochastic as well as deterministic search procedures.

Theorem 1. *Let $\mathcal{H}$ be a finite class of ranking functions. For any target ranking function $r \in \mathcal{H}$, and any set of $m = \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$ independent runs of a rank-based search procedure $\mathcal{P}$ guided by r drawn from a target distribution over inputs, there is a $1 - \delta$ probability that every $\hat{r} \in \mathcal{H}$*

that is consistent with the runs satisfies $L(\hat{r}) \leq L(r) + 2\epsilon L_{max}$*, where* L_{max} *is the maximum possible loss of any output and* $L(t)$ *is the expected loss of running the search procedure* $\mathcal{P}$ *with the ranking function* $t \in \mathcal{H}$*.*

Although the theoretical result assumes that the target cost function r is in the hypothesis space, in practice this is not guaranteed as the set of generated constraints might be quite large and diverse. To help reduce the complexity of the learning problem, in practice we only learn from a smaller set of pair-wise ranking decisions that are sufficient (see below) to preserve the best output that is encountered during search at any time step.

3.3.3 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 cost function that is consistent with the constraints causes the search to follow the same trajectory (sequence of states) and retains the best loss output that is encountered during search so far for input x. 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 three 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 $\mathcal{C}$-value) than those that are pruned.

3. *Anytime* constraints, which ensure that the search node corresponding to the best loss output that is encountered during search so far is ranked better than every other node that is uncovered by the search procedure.

Below, we will illustrate these constraints concretely for greedy search and best-first beam search noting that similar formulations for other rank-based search procedures is straightforward.

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

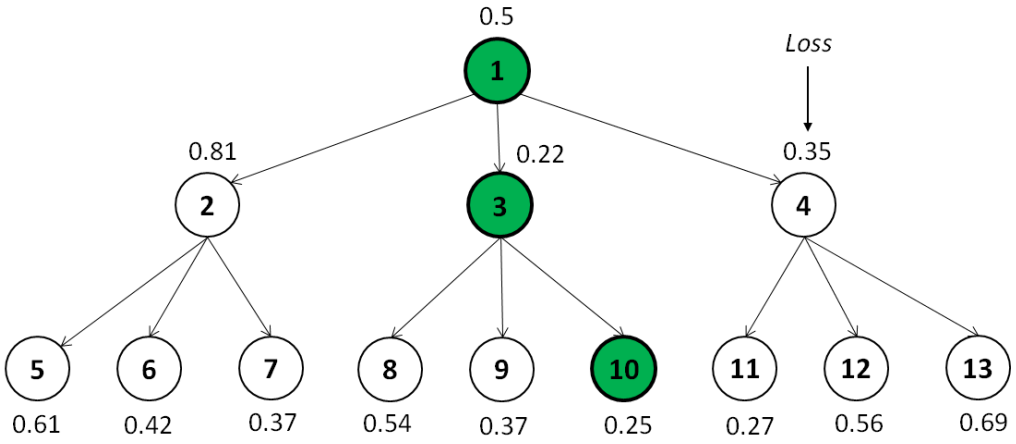

Figure 3.6: An example search tree that illustrates greedy search with loss function. Each node represents a complete input-output pair and can be evaluated using the loss function. The highlighted nodes correspond to the trajectory of greedy search guided by the loss function.

looks best according to the cost function $\mathcal{C}$ (loss function L during training). In particular, if s_i is the search state at step i, greedy search selects $s_{i+1} = \arg min_{s \in S(s_i)} \mathcal{C}(s)$, 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. Additionally, it keeps track of the best node s_i^{best} uncovered so far as evaluated by the cost function.

Let (x, y_i) correspond to the input-output pair associated[3] 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, that is, $C_{i+1} = S(s_i)$. Let s_{i+1} be the best node in the candidate set C_{i+1} as evaluated by the loss function, that is, $s_{i+1} = \arg min_{s \in C_{i+1}} L(s)$. 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 $\mathcal{C}(x, y_{i+1}) < \mathcal{C}(x, y)$. As part of the anytime constraints, we introduce a constraint between (x, y_i^{best}) and (x, y_{i+1}) according to their losses. For example, if $L(x, y_{i+1}, y^*) < L(x, y_i^{best}, y^*)$, we introduce a ranking constraint such that $\mathcal{C}(x, y_{i+1}) < \mathcal{C}(x, y_i^{best})$ and vice

[3]We use input-output pair (x, y_i) and state s_i inter-changeably for the sake of brevity.

versa.

We will now illustrate these ranking constraints through an example. Figure 4.4 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 first search step, $\{\mathcal{C}(3) < \mathcal{C}(2), \mathcal{C}(3) < \mathcal{C}(4)\}$, and $\{\mathcal{C}(3) < \mathcal{C}(1)\}$ are the pruning and anytime constraints respectively. Similarly, $\{\mathcal{C}(10) < \mathcal{C}(8), \mathcal{C}(10) < \mathcal{C}(9)\}$, and $\{\mathcal{C}(3) < \mathcal{C}(10)\}$ form the pruning and anytime constraints at second search step. Therefore, the aggregate set of constraints needed to imitate the greedy search behavior shown in Figure 4.4 are:
$\{\mathcal{C}(3) < \mathcal{C}(2), \mathcal{C}(3) < \mathcal{C}(4), \mathcal{C}(3) < \mathcal{C}(1), \mathcal{C}(10) < \mathcal{C}(8), \mathcal{C}(10) < \mathcal{C}(9), \mathcal{C}(3) < \mathcal{C}(10)\}$.

Best-first Beam Search: This is a more sophisticated search procedure compared to greedy search. Best-first beam search maintains a set of b open nodes B_i in its internal memory at every search step i, where b is the beam width. Greedy search is a special case of beam search, where beam width b equals 1. For a given input x, it traverses the search space by expanding the best node s_i in the current beam B_i (i.e., $s_i = \arg min_{s \in B_i} \mathcal{C}(s)$) and computes the next beam B_{i+1} by retaining the best b open nodes after expanding s_i, where $B_0 = \{I(x)\}$.

Best-first beam search selects the best node s_i at every search step i from its beam B_i for expansion, that is, $s_i = \arg min_{s \in B_i} L(s)$. Therefore, selection constraints need to ensure that s_i is ranked better than all the other nodes in beam B_i. Therefore, we include one ranking constraint for every node $(x, y) \in B_i \setminus (x, y_i)$ such that $\mathcal{C}(x, y_i) < \mathcal{C}(x, y)$. Let C_{i+1} be the candidate set after expanding state s_i, that is, $C_{i+1} = S(s_i) \cup B_i \setminus s_i$ and let B_{i+1} be the best b nodes in the candidate set according to the loss function. As best-first beam search prunes all the nodes in the candidate set other than those in B_{i+1}, pruning constraints need to ensure that every node in B_{i+1} is ranked better than every node in $C_{i+1} \setminus B_{i+1}$. Therefore, we generate one ranking example for every pair of nodes $((x, y_b), (x, y)) \in B_{i+1} \times C_{i+1} \setminus B_{i+1}$, requiring that $\mathcal{C}(x, y_b) < \mathcal{C}(x, y)$. Similar to greedy search, as part of the anytime constraints, we introduce a ranking constraint between (x, y_i^{best}) and (x, y_{i+1}) according to their losses.

We will now illustrate the above ranking constraints for best-first beam search through the example search tree in Figure 4.4. Let us consider best-first beam search with beam width $b = 2$ and let (B_0, B_1, B_2) correspond to the beam trajectory of search with loss function that needs to be imitated by our learner, where $B_0 = \{1\}$, $B_1 = \{3, 4\}$ and $B_2 = \{4, 10\}$. At first search step, there are no selection constraints as B_0 contains only a single node, and $\{\mathcal{C}(3) < \mathcal{C}(2), \mathcal{C}(4) < \mathcal{C}(2)\}$ and $\{\mathcal{C}(3) < \mathcal{C}(1)\}$ are the pruning and anytime constraints respec-

tively. Similarly, $\{\mathcal{C}(3) < \mathcal{C}(4)\}$, $\{\mathcal{C}(4) < \mathcal{C}(8), \mathcal{C}(4) < \mathcal{C}(9), \mathcal{C}(10) < \mathcal{C}(8), \mathcal{C}(10) < \mathcal{C}(9)\}$ and $\{\mathcal{C}(3) < \mathcal{C}(10)\}$ form the selection, pruning and anytime constraints at second search step.

The only thing that remains to be explained in Algorithm 2 is, how to learn a cost function $\mathcal{C}$ from the aggregate set of ranking examples $\mathcal{R}$ (step 15). Below we describe this rank learning procedure.

3.3.4 Rank Learner

We can use any off-the-shelf rank-learning algorithm (e.g., Perceptron, SVM-Rank) as our base learner to learn the cost function from the set of ranking examples $\mathcal{R}$. In our specific implementation we employed the online Passive-Aggressive (PA) algorithm (Crammer et al., 2006) as our base learner.[4] 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 in $\mathcal{R}$ is of the form $\mathcal{C}(x, y_1) < \mathcal{C}(x, y_2)$, 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 et al., 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 cost function. If there is a ranking error, that is, $w_t \cdot \Phi(x, y_2) - w_t \cdot \Phi(x, y_1) < \sqrt{\Delta}$, the new weights w_{t+1} that corrects the error can be obtained using the following equation.[5]

[4] In the conference version of this work, we employed the perceptron learner and followed a training approach that slightly differs from Algorithm 2. Specifically, the ranking examples for exact imitation were generated until reaching y^*, the correct output, and after that we only generate training examples to rank y^* higher than the best cost open node(s) as evaluated by the current cost function and continue the search guided by the cost function. This particular training approach may be beneficial (results in the conference paper are slightly better than those presented in this article and in practice, breaking ties via cost function is better than employing a random tie-breaker), but it is computationally very expensive (requires the ranking examples to be generated on-the-fly during each iteration of online training). However, this training methodology can be both beneficial and practical when applied on sparse search spaces.

[5] Crammer et al. (2006) prove bounds on the cumulative squared loss and therefore, they employ this particular margin constraint with $\sqrt{\Delta}$.

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

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 et al., 2005) (See Equation 7).

3.3.5 Summary of Overall Training Approach

Our search-based framework thus consists of two main learning components: 1) the search space learner, and 2) the cost function learner. We train them sequentially. First, we train the recurrent classifier as described in Section 3.2.1, which is used to define either the LDS or flipbit search spaces (see Section 3.2). Second, we train the cost function $\mathcal{C}$ to score the outputs for a given combination of search space over complete outputs S_o and a search procedure $\mathcal{A}$ as described in Algorithm 2. More specifically, for every training example (x, y^*), we run the search procedure $\mathcal{A}$ on the search space S_o instantiated for input x, using the loss function L for the specified time bound τ, and generate imitation training data (ranking examples) for the cost function learning (see Section 3.3.3). We give the aggregate set of imitation training data to a rank learner to train the cost function $\mathcal{C}$ as described in Section 3.3.4.

3.4 Empirical Results

In this section we empirically investigate our approach along several dimensions and compare it against the state-of-the-art in structured prediction.

3.4.1 Experimental Setup

We evaluate our approach on the following six structured prediction problems including five 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 data set contains roughly 6600 examples divided into 10 folds (Taskar et al., 2003). We consider two different variants of this task as in Hal Daumé III et al. (2009). For the `HW-Small` version of the problem,, we employ 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 (Sejnowski and Rosenberg, 1987). There are 1000 training words and 1000 test words in the standard data set. 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 a word.

- **Chunking.** The goal in this task is to syntactically chunk English sentences into meaningful segments. We consider the full syntactic chunking task and use the data set from the CONLL 2000 shared task,[6] which consists of 8936 sentences of training data and 2012 sentences of test data.

- **POS tagging.** We consider the tagging problem for the English language, where the goal is to assign the part-of-speech tag to each word in a sentence. The standard data from Wall Street Journal (WSJ) corpus[7] was used in our experiments.

- **Scene labeling.** This data set 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 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.

We used F_1 loss as the loss function for the chunking task and employed Hamming loss for all other tasks.

[6] CONLL task can be found at `http://www.cnts.ua.ac.be/conll2000/chunking/`.
[7] WSJ corpus can be found at `http://www.cis.upenn.edu/~treebank/`.

For all sequence labeling problems, the recurrent classifier labels a sequence using a left-to-right ordering and for scene labeling uses an ordering of top-left to right-bottom in a row-wise raster form. To train the recurrent classifiers, the output label of the previous token is used as a feature to predict the label of the current token for all sequence labeling problems with the exception of chunking and POS tagging, where labels of the two previous tokens were used. For scene labeling, the labels of neighborhood (top and left) patches were used. In all our experiments, we train the recurrent classifier using exact imitation (see Section 3.2) with the Perceptron algorithm for 100 iterations with a learning rate of 1.

Unless otherwise indicated, the cost functions learned over input-output pairs are second order, meaning that they have features over neighboring label pairs and triples along with features of the structured input. For the scene labeling task, we consider pairs and triples along both horizontal and vertical directions. We trained the cost function via exact imitation as described in Section 3.3 using 50 iterations of Passive-Aggressive training.

3.4.2 Comparison to State-of-the-Art

We experimented with several instantiations of our framework. First, we consider our framework using a greedy search procedure for both the LDS and flipbit spaces, denoted by **LDS-Greedy** and **FB-Greedy**. Unless otherwise noted, in both training and testing, the greedy search was run for a number of steps equal to the length of the sequence. Using longer runs did not impact results significantly. Second, we performed experiments with best-first beam search for different beam widths and search steps, but we didn't see significant improvements over the results with greedy search. Therefore, we do not report these results. Third, to see the impact of adding additional search at test time to a greedily trained cost function, we also used the cost function learned by LDS-Greedy and FB-Greedy in the context of a best-first beam search (beam width = 100) at test time in both the LDS and flipbit spaces, denoted by **LDS-BST(greedy)** and **FB-BST(greedy)**. We also report the performance of using our trained recurrent classifier (**Recurrent**) to make predictions, which is equivalent to performing no search since both search spaces are initialized to the recurrent classifier output. Finally, we also report the exact imitation accuracy ($100 * (1 - \epsilon_{ei})$), which as described earlier (see Section 3.2) is the accuracy being directly optimized by the recurrent classifier and is related to the structure of the flipbit and LDS spaces.

We compare our results with other structured prediction algorithms including **CRFs** (Lafferty et al., 2001), **SVM-Struct** (Tsochantaridis et al., 2004), **SEARN** (Hal Daumé III et al.,

ALGORITHMS	DATA SETS						
	HW-Small	HW-Large	Stress	Phoneme	Chunk	POS	Scene labeling
a. Comparison to state-of-the-art							
$100 * (1 - \epsilon_{ei})$	73.9	83.99	77.97	77.09	88.84	92.5	**78.61**
Recurrent	65.67	74.87	72.82	73.58	88.51	92.15	56.64
LDS-Greedy	82.59	92.59	78.85	79.09	94.62	**96.93**	72.95
FB-Greedy	80.3	89.38	77.93	78.43	93.96	96.87	67.67
CRF	80.03	86.89	78.52	78.91	**94.77**	96.84	-
SVM-Struct	80.36	87.51	77.99	78.3	93.64	96.81	-
SEARN	82.12^{B}	90.58^{B}	76.15	77.26	94.44^{B}	95.83	62.31
CASCADES(2012)	69.62	87.95	77.18	69.77	-	96.82	-
CASCADES(updated)	**86.98**	**96.78**	**79.59**	**82.44**	-	96.82	-
b. Results with Additional Search							
LDS-BST(greedy)	$\mathbf{83.81}^{+}$	$\mathbf{93.17}^{+}$	78.76	78.87	94.63	96.95	$\mathbf{74.12}^{+}$
FB-BST(greedy)	81.19^{+}	90.21^{+}	77.61	78.32	93.98	96.91	69.23^{+}
c. Results with DAgger							
LDS-Greedy	$\mathbf{83.62}^{+}$	$\mathbf{93.24}^{+}$	$\mathbf{79.81}^{+}$	$\mathbf{79.97}^{+}$	94.61	96.91	$\mathbf{74.27}^{+}$
FB-Greedy	81.28^{+}	90.45^{+}	78.96^{+}	79.23^{+}	93.94	96.89	69.63^{+}
d. Results with Third-Order Features							
LDS-Greedy	85.85^{+}	95.08^{+}	80.21^{+}	81.61^{+}	94.63	96.97	$\mathbf{74.71}^{+}$
FB-Greedy	83.18^{+}	92.66^{+}	79.23^{+}	80.65^{+}	94.17^{+}	96.94	69.81^{+}
CASCADES(2012)	81.87^{+}	93.76^{+}	73.48	68.98	-	96.84	-
CASCADES(updated)	$\mathbf{89.18}^{+}$	$\mathbf{97.84}^{+}$	$\mathbf{80.49}^{+}$	$\mathbf{82.59}^{+}$	-	96.84	-

Table 3.1: Prediction accuracy results of different structured prediction algorithms and variations of our framework. A + indicates that the particular variation being considered resulted in improvement.

2009) and **CASCADES** (Weiss and Taskar, 2010). For these algorithms, we report the best published results whenever available. In the remaining cases, we used publicly available code or our own implementation to generate those results. Ten percent of the training data was used to tune the hyper-parameters. CRFs were trained using SGD.[8] SVM^{hmm} was used to train SVM-Struct and the value of the parameter C was chosen from $\left\{10^{-4}, 10^{-3}, \cdots, 10^{3}, 10^{4}\right\}$ based on the validation set. Cascades were trained using the implementation[9] provided by the authors, which can be used for sequence labeling problems with Hamming loss. We present two different results for CASCADES: 1) CASCADES(2012) employs the version of the code at the original time this work was done, 2) CASCADES(updated) employs the most recent updated[10] version of the CASCADES training code, which significantly improves on the CASCADES(2012). For SEARN we report the best published results with a linear classifier (i.e., linear SVMs instead of Perceptron) as indicated by B in the table and otherwise ran our own implementation of SEARN with optimal approximation as described in Hal Daumé III et al. (2009) and optimized the interpolation parameter β over the validation set. Note that we do not compare our results to SampleRank due to the fact that its performance is highly dependent on the hand-designed proposal distribution, which varies from one domain to another.

Table 3.1a shows the prediction accuracies of different algorithms ('-' indicates that we were not able to generate results for those cases as the software package was not directly applicable). Across all benchmarks we see that the most basic instantiations of our framework, LDS-Greedy and FB-Greedy, produce results that are comparable or significantly better than all the other methods excluding[11] CASCADES(updated). This is particularly interesting, since these results are achieved using a relatively small amount of search and the simplest search method, and results tend to be the same or better for our other instantiations. A likely reason that we are outperforming CRFs and SVM-Struct is that we use second-order features while those approaches use first-order features, since exact inference with higher order features is too costly, especially during training. As stated earlier, one of the advantages of our approach is that we can use higher-order features with negligible overhead.

Finally, the improvement in the scene labeling domain is the most significant, where SEARN

[8] SGD code can be found at `http://leon.bottou.org/projects/sgd`.

[9] Cascades code can be found at `http://code.google.com/p/structured-cascades/`.

[10] Most recent based on personal communication with the author.

[11] A followup work (Doppa et al., 2014a) that employs two distinct functions for guiding the search and scoring the candidate outputs generated during search performs comparably or better than CASCADES(updated) across all benchmarks.

achieves an accuracy of 62.31 versus 72.95 for LDS-Greedy. In this domain, most prior work has considered the simpler task of classifying entire images into one of a set of discrete classes, but to the best of our knowledge no one has considered a structured prediction approach for patch classification. The only reported result for patch classification that we are aware of Vogel and Schiele (2007) obtains an accuracy of 71.7 (versus our best performance of 74.27) with non-linear SVMs trained i.i.d. on patches using more sophisticated features than ours.

3.4.3 Framework Variations

Adding More Search. Table 3.1b shows that LDS-BST(greedy) and FB-BST(greedy) are generally the same or better than LDS-Greedy and FB-Greedy, with the biggest improvements in handwriting recognition task and the challenging scene labeling ('+' indicates improvement). Results improve from 82.59 to 83.81 in HW-Small, from 92.59 to 93.17 in HW-Large and from 72.95 to 74.12 in the scene labeling task. This shows that it can be an effective strategy to train using greedy search and then insert that cost function into a more elaborate search at test time for further improvement. As noted earlier, in the domains we considered, training with a more sophisticated search procedure like beam search did not improve results over greedy search. This demonstrates the efficiency of our search spaces and can be considered as a positive result.

Exact Imitation vs. DAGGER**.** Our experiments show that the simple exact imitation approach for cost function training performs extremely well on our problems. However, cost functions trained via exact imitation can be prone to error propagation (Kääriäinen, 2006; Ross and Bagnell, 2010). It is interesting to consider whether addressing this issue might improve results further. Therefore, we experimented with DAGGER (Ross et al., 2011), a more advanced imitation training regime that addresses error propagation through on-line training and expert demonstrations. At a high-level, DAGGER learns on-line from an aggregate data set collected over several iterations. The first iteration corresponds to the data produced by exact imitation of the expert. Further iterations correspond to the actions suggested by the expert on trajectories produced by a mixture of the learned policy from the previous iteration and the expert policy. This allows DAGGER to learn from states visited by its possibly erroneous learned policy and correct its mistakes using expert input. In our adaptation, the "learned policy" corresponds to the decisions made by the greedy search guided by the cost function as the heuristic, and the "expert policy" corresponds to the decisions made by greedy search guided by the loss function. 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 setting in our DAGGER experiments.

We picked the best cost function based on a validation set after 5 iterations of DAGGER, noting that no noticeable improvement was observed after 5 iterations. Table 3.1c shows the results of LDS-Greedy and FB-Greedy obtained by training with DAGGER. We see that there are some improvements over the cost function trained with exact imitation, although the improvements are quite small ('+' indicates improvement). As we will show later, we get much more positive results for DAGGER in the context of pruned search spaces.

Higher-Order Features. One of the advantages of our framework compared to other approaches for structured prediction is the ability to use more expressive feature spaces with negligible computational overhead. Table 3.1d shows results using third-order features (compared to second-order results in Table 3.1a) for LDS-Greedy, FB-Greedy and Cascades.[12] Note that it is not practical to run the other methods (e.g., CRFs and SVM-Struct) using third-order features due to the substantial increase in inference time. The results of LDS-Greedy and FB-Greedy with third-order features improve over the corresponding results with second-order features across the board ('+' indicates improvement). Finally, we note that while CASCADES(updated) is able to improve performance by using third-order features, the improvement is negligible for phoneme prediction.

LDS space vs. Flipbit space. We see that generally the instances of our method that use the LDS space outperform the corresponding instances that use the Flipbit space. Interestingly, if there is a large difference between the exact imitation accuracy $1-\epsilon_{ei}$ and the recurrent classifier accuracy (e.g., Handwriting and Scene labeling), then the LDS space is significantly better than the flip-bit space. This is particularly true in our most complex problem of scene labeling where this difference is quite large, as is the gap between LDS and Flipbit. These results show the benefit of using the LDS space and empirically confirm our observations in Section 3.2 that the quality of the LDS and Flipbit spaces are related to the exact imitation and recurrent error rates respectively.

3.4.4 Results with Sparse Search Spaces

Recall that sparse search spaces are parameterized by k, the sparsity parameter (see Section 3.2.5). Small values of k lead to proportionately smaller branching factors for search. We perform ex-

[12] Cascades code can be found at `http://code.google.com/p/structured-cascades/`.

periments for different values of k to evaluate the effectiveness of sparse search spaces (i.e., LDS-k and FB-k). For example, **LDS-2** and **FB-2** correspond to the configurations where k equals 2. We only report the results for $k = 2$ and $k = 4$ noting that we didn't see major improvements for larger values of k. For all these experiments, we run greedy search for a number of steps equal to the length of the sequence during both training and testing. For greedy search, the computation time can be expected to be linearly related to k since the main computational bottleneck is the generation of $T \cdot k$ successors for each node encountered during the search.

Results of Cost Function Trained on Complete Search Spaces. Table 3.2b gives the results of using a cost function trained on a complete (non-sparse) search space (as in the previous experiments) to make predictions via the pruned spaces. As we can see, the gap between the results of LDS-2 and FB-2 and the corresponding results obtained using complete search space (see Table 3.2a) is very small. This means that we get huge speed improvements during testing with only a small loss in accuracy (more details on speedup below). The accuracy loss reduces with less sparse search spaces (LDS-4 and FB-4), but comes at the expense of more computation time.

Results of Cost Function Trained on Sparse Search Spaces. It is natural to expect that performance on sparse search spaces might improve if the cost function is trained using the same sparse search space. Further, since conducting searches in the sparse spaces is computationally cheaper, learning directly in sparse spaces can be much more efficient. Table 3.2c shows the results of training the cost function on the sparse search spaces using exact imitation. As we can see, accuracies of LDS-2 and FB-2 slightly degrade compared to the corresponding results in Table 3.2b, but the results of LDS-4 and FB-4 equal or slightly improve in almost all cases except for scene labeling. These results show that we get speed improvements during both training and testing with little loss in accuracy.

Contrary to expectation, the above results show that when using the LDS-2 and FB-2 spaces, training directly on those spaces was often slightly worse than training on the complete spaces. One hypothesis for this observation is that the number and variation of states encountered during training by the exact imitation approach is much less for sparser spaces. This can possibly hurt robustness of the learned cost function. This suggests that a more sophisticated approach such as DAGGER might be more effective since it effectively generates a wider diversity of states during training.

Table 3.2 shows the results of training with DAGGER, which confirm the above hypothesis. First, DAGGER significantly improves over the results obtained with exact imitation (see

ALGORITHMS	DATA SETS						
	HW-Small	HW-Large	Stress	Phoneme	Chunk	POS	Scene labeling
a. Accuracy results of training and testing on complete search space							
LDS	**82.59**	**92.59**	**78.85**	**79.09**	**94.62**	96.93	**72.95**
FB	80.3	89.38	77.93	78.43	93.96	96.87	67.67
b. Accuracy results of cost function trained on complete search space							
LDS-2	81.02	91.38	78.62	79.79	93.95	96.13	69.87
FB-2	79.47	86.95	77.82	79.23	93.18	96.08	65.43
LDS-4	**82.55**	**92.45**	**78.85**	**79.97**	**94.55**	96.77	**72.11**
FB-4	80.43	88.40	77.93	79.23	94.23	96.81	66.98
c. Accuracy results of cost function trained on sparse search space via Exact Imitation							
LDS-2	80.17	90.43	78.72	78.90	94.08	96.29	68.62
FB-2	78.95	87.61	77.57	78.80	94.11	96.45	63.45
LDS-4	**83.12**	**92.84**	**78.85**	79.46	94.55	96.51	**70.69**
FB-4	80.80	90.04	77.93	**79.59**	94.39	96.57	65.67
d. Accuracy results of cost function trained on sparse search space via DAgger							
LDS-2	82.54	92.14	79.27	80.57	94.27	96.38	71.56
FB-2	80.73	89.65	78.94	80.48	94.32	96.55	66.78
LDS-4	**85.53**	**94.14**	**79.81**	**81.23**	94.58	96.85	**73.61**
FB-4	82.87	91.75	78.96	81.26	94.56	96.89	68.71
e. Timing results (avg. time per greedy search step in milli seconds)							
LDS	40.0	40.0	1.0	23.0	421.0	695.0	2660.0
FB	20.0	19.0	1.0	10.0	134.0	170.0	1740.0
LDS-2	3.0	3.8	0.7	1.0	70.0	65.0	580.0
FB-2	2.0	2.0	0.6	0.7	27.0	17.0	350.0
LDS-4	7.0	7.2	1.3	2.0	160.0	140.0	1350.0
FB-4	3.6	3.6	1.2	1.3	60.0	35.0	790.0

Table 3.2: Prediction accuracy and timing results for greedy search comparing sparse and complete search spaces.

Table 3.2c) across the board. Second, the results of training (via DAGGER) and testing on sparse search spaces are better than the results of training on complete search spaces and testing on sparse search spaces (see Table 3.2b). This agrees with the intuition that training on the search space used for testing should be superior to training on a different search space. Third, the results of LDS-4 and FB-4 with DAGGER are significantly better than the results obtained by training and testing on complete search spaces (see Table 3.2a). This indicates that training on sparse search spaces via DAgger is very effective and gives us speed improvement with no loss in accuracy and sometimes improves accuracy compared to the complete spaces.

Inference Time and Anytime Performance. Table 3.2e shows the timing results (avg. time per greedy search step in secs.) of our approach during testing using sparse (LDS-k and FB-k) and complete search spaces (LDS and FB). As we can see, we get speed improvement by a factor of ten (roughly) with sparse search spaces. Note that the speedup will generally be larger for problems with larger numbers of labels L, since the number of successors decreases from $T \cdot (L-1)$ to $T \cdot k$. We would like to point out that sparse search spaces will also improve the training time of our approach (fewer ranking examples in step 11 of Algorithm 2), and can be advantageous[13] compared to standard approaches including CRFs and SVM-Struct. Further, we compare the anytime curves of configurations of our approach with sparse and complete search spaces, which show the accuracy achieved by a method versus an inference time bound at prediction time. Figure 3.7 shows the anytime curves for all the problems except stress prediction, where there is hardly any difference due to the small label set size (5 labels). Note that all these results are for training with DAgger.

From the anytime curves, it is clear that the configurations with sparse search spaces have a much better anytime profile compared to the ones with complete search spaces. LDS-2 and FB-2 reach the respective accuracies of LDS and FB very quickly in all the cases except for POS and Scene labeling, where there is a small loss in accuracy. However, LDS-4 and FB-4 recover the accuracy losses in those two cases. These results demonstrate that sparse search spaces would be highly effective in those situations, where there is a need to make anytime predictions.

Comparing the anytime curves of LDS and FB, we can see that LDS is comparable or better than FB in all cases other than Chunking and POS.[14] This is especially true for the handwriting recognition and scene labeling problems. In the anytime curves for scene labeling task, we can see that LDS is dominant and improves accuracy much more quickly than FB. For example,

[13] It is hard to do a fair comparison of wall clock times due to differences in implementations.

[14] The experimental setup only differs in the search space (LDS or FB) employed during training and testing.

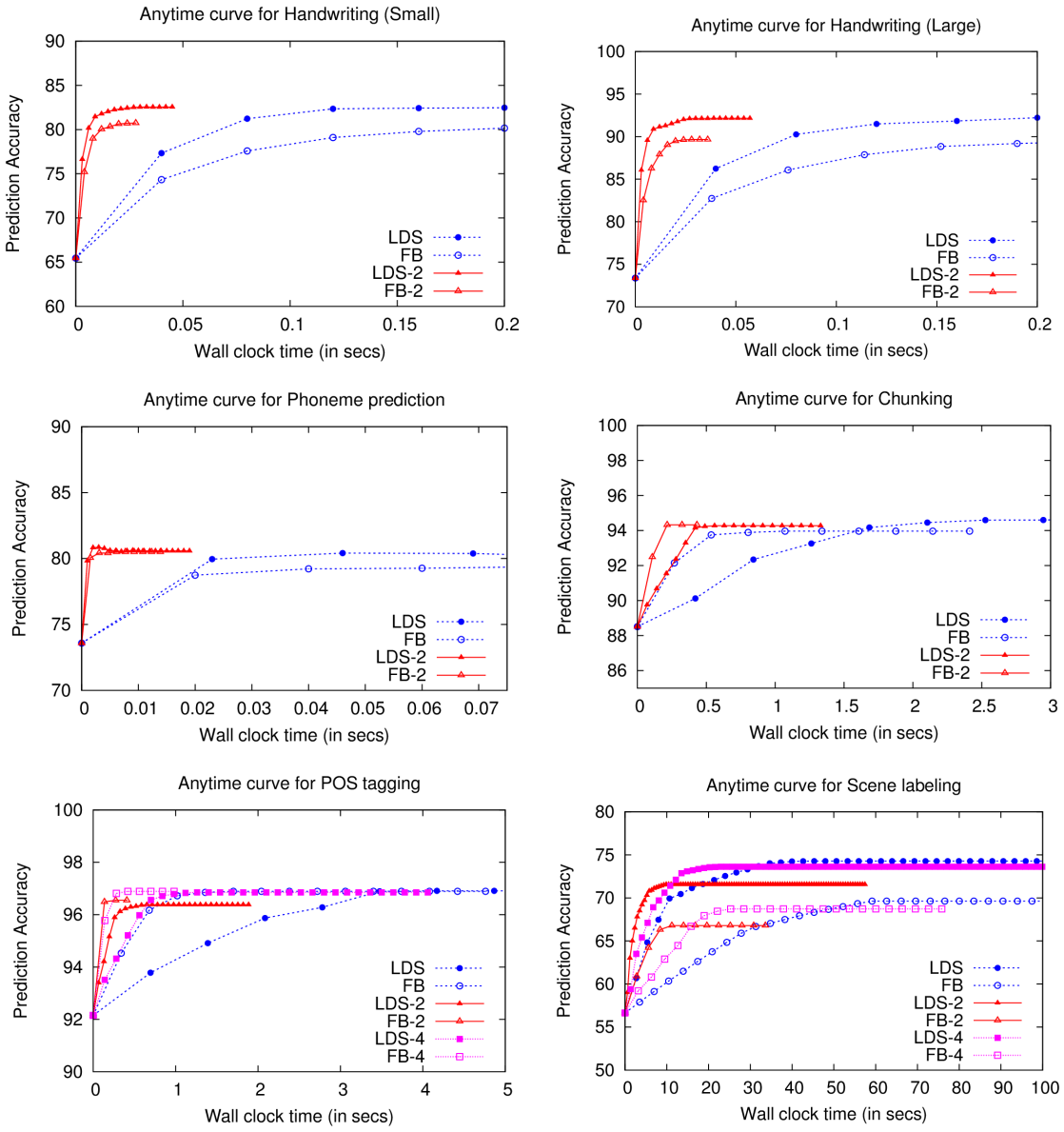

Figure 3.7: Anytime curves for greedy search comparing sparse and complete search spaces.

a 10 second time bound for LDS achieves the same accuracy as FB using 70 seconds. This shows the benefit of using the LDS space. In the case of Chunking and POS, there is almost no difference between the accuracy of the recurrent classifier and exact imitation accuracy (see Table 3.1a), so LDS does not provide any extra benefit over FB. Recall that there is significant additional overhead for successor generation for LDS compared to flipbit. To generate each successor, LDS must evaluate the recurrent classifier at sequence positions after discrepancies are introduced. On the other hand, the flipbit space need not evaluate the recurrent classifier during successor generation, but only uses the recurrent classifier to generate the initial state. In Chunking and POS, the additional overhead of the LDS search does not payoff in improved accuracy and the anytime curve of flipbit is accordingly better. All these findings are true for the sparser versions of LDS and FB as well.

3.5 Summary

We studied a general approach for structured prediction based on search in the space of complete outputs. We showed how powerful classifiers can be leveraged to define an effective search space over complete outputs called limited discrepancy search (LDS) space, and gave a generic cost function learning approach to score the outputs for any given combination of search space and search strategy. Our experimental results showed that a very small amount of search in the LDS space is needed to improve upon the state-of-the-art performance, validating the effectiveness of our framework. We also addressed some of the scalability issues via a simple pruning strategy that creates sparse search spaces that are more efficient to search in.

Chapter 4: $\mathcal{HC}$-Search Framework

In this chapter, 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 existing output space search approaches have achieved state-of-the-art performance on a number of benchmark problems (see Chapter 3 and (Wick et al., 2011)), a primary contribution of this chapter 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.

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.

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 chapter proceeds as follows. In Section 4.1, 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 4.2. Section 4.3 presents our experimental results followed by an engineering methodology for applying our framework to new problems in Section 4.4. Finally, Section 4.5 provides a summary of what we learned from this work.

4.1 $\mathcal{HC}$-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.

4.1.1 Problem Setup

Recall that 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.

4.1.2 Search Spaces and Search Strategies

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 3.3 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. Recall that we can quantify the quality of a search space, independently of the specific search strategy, by considering the expected depth of target outputs y^*. 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 simple flipbit space, the Limited Discrepancy Search (LDS) space, 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, which has been shown to effectively uncover high-quality outputs at relatively shallow search depths (see Chapter 3).

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 (see Chapter 3).

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 on the quality of the search heuristic $\mathcal{H}$. Prior work (Chapter 3 and (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.

4.1.3 $\mathcal{HC}$-Search Approach

Our approach is parameterized by a search space over complete outputs $\mathcal{S}_l$ (e.g., LDS 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 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 return the least-cost output $\hat{y}$ that is generated during the search as the prediction for input x. Figure 4.1 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.,

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

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

$$\mathcal{E}\left(\mathcal{H}, \mathcal{C}\right) = \mathbb{E}_{(x,y^*)\sim\mathcal{D}} \; L\left(x, \hat{y}, y^*\right) \tag{4.1}$$

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.,

$$(\mathcal{H}^*, \mathcal{C}^*) = \arg\min_{(\mathcal{H},\mathcal{C}) \in \mathbf{H} \times \mathbf{C}} \; \mathcal{E}\left(\mathcal{H}, \mathcal{C}\right) \tag{4.2}$$

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 2. *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 $\min_{(\mathcal{H},\mathcal{C})} \mathcal{E}(\mathcal{H}, \mathcal{C})$ even when using the same feature space.*

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.

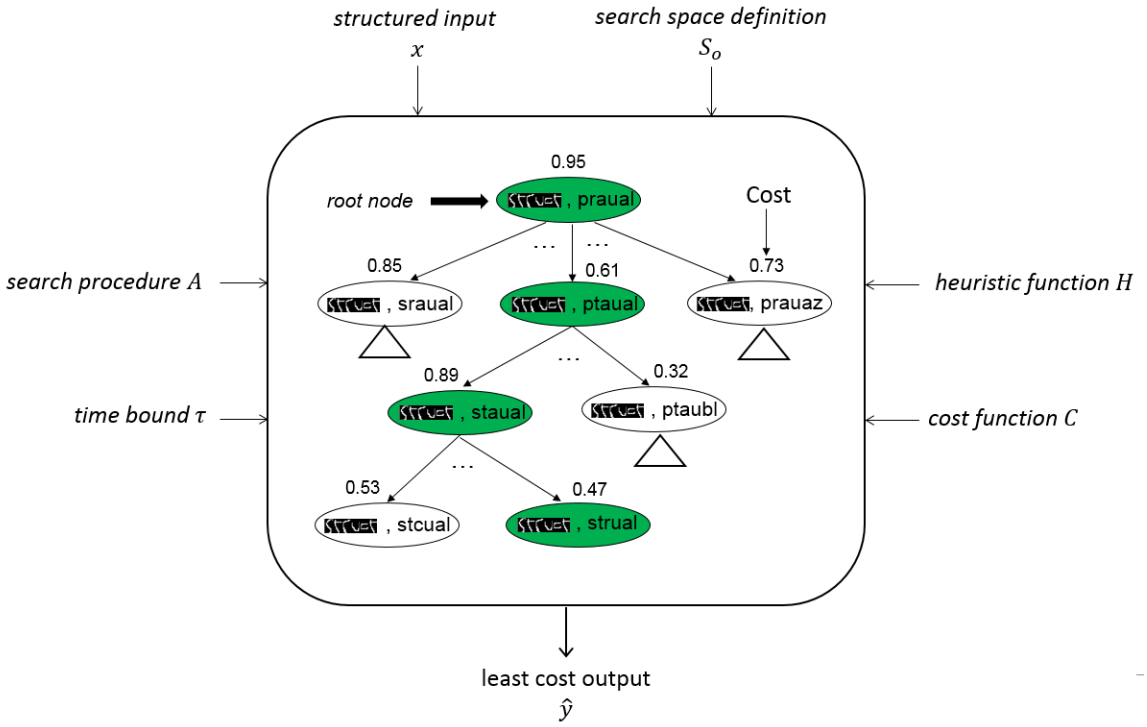

Figure 4.1: A high level overview of our $\mathcal{HC}$-Search framework. Given a structured input x and a search space definition S_o, we first instantiate a search space over complete outputs. Each search node in this space consists of a complete input-output pair. Next, we run a search procedure $\mathcal{A}$ (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.

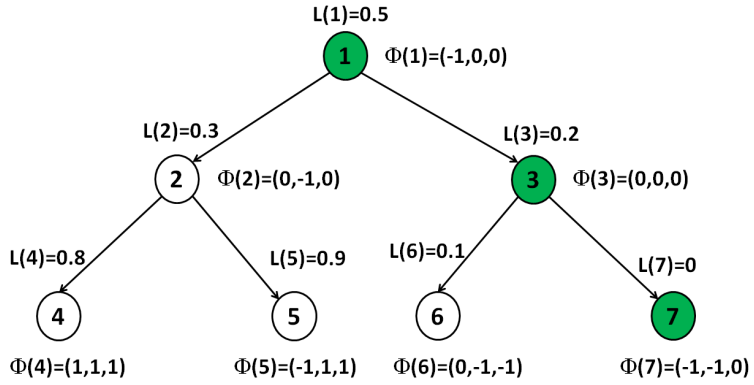

Figure 4.2: An example that illustrates that $\mathcal{C}$-Search can suffer arbitrarily large loss compared to $\mathcal{HC}$-Search.

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.

4.1.4 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 et al., 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 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.

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 et al. (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 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 effec-

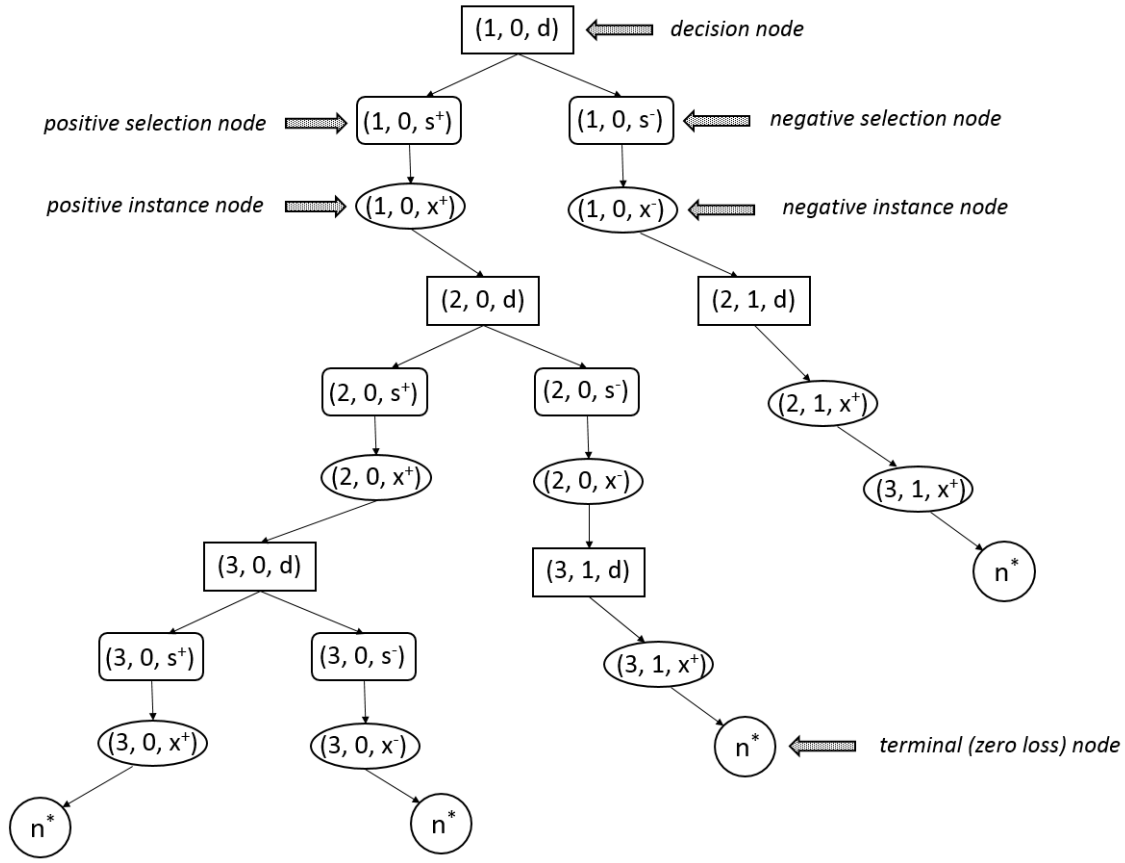

Figure 4.3: An example search space for $T = \{x_1, x_2, x_3\}$ and $k = 1$. All greedy paths terminate at the zero loss node n^* and no path selects more than one instance to include in the mistake set T^-.

tively 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 4.3 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^-. □

4.2 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 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.

4.2.1 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:

$$\mathcal{E}\left(\mathcal{H},\mathcal{C}\right) = \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x, y^*_{\mathcal{H}}, y^*\right)}_{\epsilon_{\mathcal{H}}} + \underbrace{\mathbb{E}_{(x,y^*)\sim\mathcal{D}}\, L\left(x,\hat{y},y^*\right) - L\left(x,y^*_{\mathcal{H}},y^*\right)}_{\epsilon_{\mathcal{C}|\mathcal{H}}} \tag{4.3}$$

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 *decoupled* in the sense that $\hat{\mathcal{H}}$ is learned without considering the implications for learning $\hat{\mathcal{C}}$. While the proof of Theorem 2 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.

4.2.2 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 and Dietterich, 1995). 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. 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.

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 learning single cost functions for output-space search discussed in the previous chapter. 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.

Algorithm 3 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 $\mathcal{H}$

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

[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).

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

Algorithm 3 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); and 5) τ_{max}, the search time bound (e.g., number of search steps).

The algorithmic description of Algorithm 3 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, 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 3 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

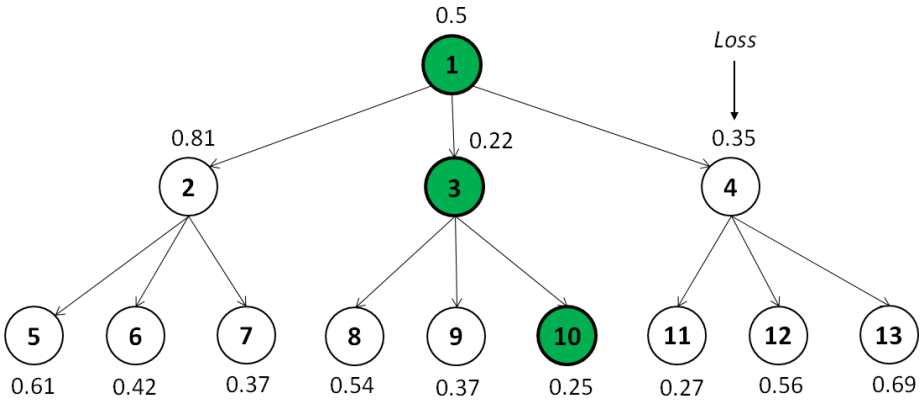

Figure 4.4: An example search tree that illustrates greedy search with loss function. Each node represents a complete input-output pair and can be evaluated using the loss function. The highlighted nodes correspond to the trajectory of greedy search guided by the loss function.

them for greedy search through a simple example.

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 previous chapter

for beam search formulation).

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 $s_{i+1} = \text{argmin}_{s \in S(s_i)} \mathcal{H}(s)$, 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.

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., $s_{i+1} = \text{argmin}_{s \in C_{i+1}} L(s)$. 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 4.4 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 4.4 are:
$\{\mathcal{H}(3) < \mathcal{H}(2), \mathcal{H}(3) < \mathcal{H}(4), \mathcal{H}(10) < \mathcal{H}(8), \mathcal{H}(10) < \mathcal{H}(9)\}$.

4.2.3 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.

We formulate the cost function training problem as an instance of *rank learning* problem (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 $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$. 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 4) 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 4 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

1: Divide the training set $\mathcal{D}$ into k folds $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k$
2: *// Learn k different heuristics $\mathcal{H}_1, \cdots, \mathcal{H}_k$*
3: **for** $i = 1$ to k **do**
4: $T_i = \cup_{j \neq i} \mathcal{D}_j$ *// training data for heuristic $\mathcal{H}_i$*
5: $\mathcal{H}_i =$ **Learn-Heuristic**$(T_i, \mathcal{S}_o, L, \mathcal{A}, \tau_{max})$ *// heuristic learning via Algorithm 3*
6: **end for**
7: *// Generate ranking examples for cost function training*
8: Intialize the set of ranking examples $\mathcal{R} = \emptyset$
9: **for** $i = 1$ to k **do**
10: **for** each training example $(x, y^*) \in \mathcal{D}_i$ **do**
11: 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**$(x, \mathcal{S}_o, \mathcal{A}, \mathcal{H}_i, \tau_{max})$
12: 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 $l_{best} = min_{y \in \mathcal{Y}_{\mathcal{H}_i}(x)} L(x, y, y^*)$
13: **for** each pair of outputs $(y_{best}, y) \in \mathcal{Y}_{best} \times \mathcal{Y}_{\mathcal{H}_i}(x) \setminus \mathcal{Y}_{best}$ **do**
14: Add ranking example $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$ to $\mathcal{R}$
15: **end for**
16: **end for**
17: **end for**
18: *// Train cost function on all the ranking examples*
19: $\mathcal{C} =$ **Rank-Learner**$(\mathcal{R})$
20: **return** learned cost function $\mathcal{C}$

Algorithm 4 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).

4.2.4 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 et al., 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 et al., 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 correct 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 et al., 2005) (See

Equation 7) in addition to the $\mathcal{C}$-Search framework of the previous chapter.

4.3 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.

4.3.1 Datasets

We evaluate our approach on the following four structured prediction problems including three benchmark sequence labeling problems and a 2D image labeling problem (same as the ones used in Chapter 3). We did not experiment with Chunking and POS tagging problems because there is hardly any room for improvement (see Chapter 3).

- **Handwriting Recognition (HW).** The input is a sequence of binary-segmented handwritten letters and the output is the corresponding character sequence $[a-z]^+$.
- **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.
- **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 and 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*).

4.3.2 Experimental Setup

For our $\mathcal{HC}$-Search experiments, we use the Limited Discrepancy Space (LDS) exactly as described in the previous chapter as our search space over structured outputs. Prior work with $\mathcal{C}$-Search has shown that greedy search works quite well for most structured prediction tasks, particularly when using the LDS space (see Chapter 3). 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 ($\Phi_{\mathcal{H}}$ and $\Phi_{\mathcal{C}}$ are same). We measure error with Hamming loss unless otherwise noted.

4.3.3 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 and Taskar, 2010) and $\mathcal{C}$**-Search**, which is identical to $\mathcal{HC}$-Search except that it uses a single-function for output space search (see Chapter 3). We also show the performance of **Recurrent**, which is a simple recurrent classifier trained exactly as in the previous chapter. The top section of Table 4.1 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 et al. (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.

[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 4.1: Error rates of different structured prediction algorithms.

4.3.4 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 4.1b 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.

4.3.5 Loss Decomposition Analysis

We now examine $\mathcal{HC}$-Search and $\mathcal{C}$-Search in terms of their loss decomposition (see Equation 4.3) 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 4.2 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.

4.3.6 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 4.2b 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, and using appropriate training data to learn each function.

Datasets	HW-Small			HW-Large			Stress			Phoneme			Scene		
Error	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\mid\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\mid\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\mid\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\mid\mathcal{H}}$	$\epsilon_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\mid\mathcal{H}}$
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 4.2: $\mathcal{HC}$-Search: Error decomposition of heuristic and cost function.

4.3.7 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 and 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 et al., 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 4.2c 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 4.2d, 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 in 4.2c. 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.

4.3.8 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 4.5 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.

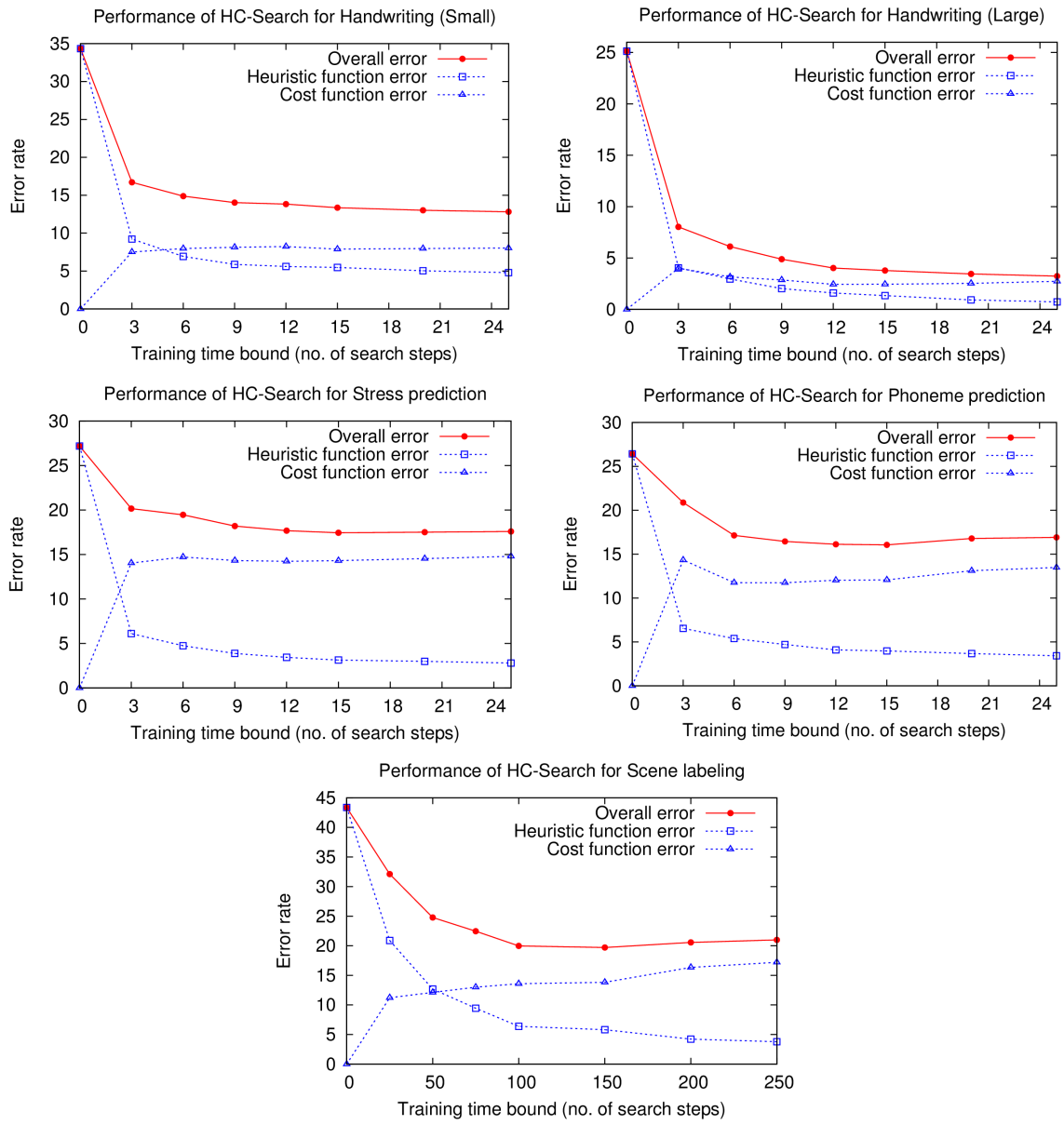

Figure 4.5: $\mathcal{HC}$-Search results for training with different time bounds. We have training time bound (i.e., no. of greedy search steps) on x-axis and error on y-axis. There are three curves in each graph corresponding to overall loss $\epsilon_{\mathcal{HC}}$, generation loss $\epsilon_{\mathcal{H}}$ and selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$.

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

Table 4.3: Results for training with non-hamming loss functions.

4.3.9 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 **??** 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 result is preliminary and more extensive evaluation is needed to generalize this result to other cost functions and domains.

4.3.10 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 (which 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. In the previous

chapter, 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. However, more work needs to be done on learning pruning rules to improve the efficiency of the $\mathcal{HC}$-Search approach.

4.4 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.

4.4.1 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 that is being employed, i.e., search depth at which target outputs can be found. In the previous chapter, we 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. 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 and Elhadad, 2010); greedy classifiers for co-reference resolution (Ma et al., 2014; Chang et al., 2013; Stoyanov and Eisner, 2012) and event extraction (Li et al., 2013); sequential labelers for boundary detection of objects in images (Payet and Todorovic, 2013); iterative classifiers for collective inference in relational networks (Sen et al., 2008; Doppa et al., 2009, 2010); classifier chains for multi-label prediction (Read et al., 2011); and greedy planners

for planning with preferences (Xu et al., 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 et al., 2014c).

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 (meaning we cannot recover from search errors), 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 (the size of the structured output) is a reasonable value for the time-bound (Figure 4.5 provides justification for this choice).

4.4.2 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.

4.5 Summary

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. We also developed an engineering methodology for applying this framework.

Chapter 5: Search-based Multi-Label Prediction

In this chapter, we adapt the $\mathcal{HC}$-Search framework to the problem of multi-label prediction, where the learner needs to predict multiple labels for a given input example (Tsoumakas et al., 2012). Multi-label problems commonly arise in domains involving data such as text, images, audio, and bio-informatics where instances can fall into overlapping conceptual categories of interest. For example, in document classification an input document can belong to multiple topics and in image classification an input image can contain multiple scene properties and objects of interest.

An important aspect of most multi-label problems is that the individual output labels are not independent, but rather are correlated in various ways. One of the challenges in multi-label prediction is to exploit this label correlation in order to improve accuracy compared to predicting labels independently. Unfortunately, existing approaches for multi-label prediction that consider label correlation suffer from the intractable problem of making optimal predictions (inference) (Dembczynski et al., 2010; Ghamrawi and McCallum, 2005; Zhang and Zhang, 2010; Guo and Gu, 2011; Petterson and Caetano, 2011). Another challenge is to automatically adapt the learning approach to the task loss function that is most appropriate for the real-world application at hand. However, most approaches are designed to minimize a single multi-label loss function (Elisseeff and Weston, 2001; Read et al., 2011; Fürnkranz et al., 2008). There are existing frameworks for multi-label prediction that can handle varying loss functions, but unfortunately they are non-trivial to adapt for a new task loss based on the needs of the application. For example, Probabilistic Classifier Chains (PCC) require a Bayes optimal inference rule (Dembczynski et al., 2010) and Structured Support Vector Machines (SSVMs) require a loss-augmented inference routine for the given task loss (Tsochantaridis et al., 2005).

In this work, we treat multi-label learning as a special case of structured-output prediction (SP), where each input x is mapped to a binary vector y (i.e., a structured output) that indicates the set of labels predicted for x. The main contribution of this chapter is to investigate a simple framework for multi-label prediction called Multi-Label Search (MLS) that makes joint predictions without suffering from intractability of the inference problem, and can be easily adapted

to optimize arbitrary loss functions[1]. The framework is a straightforward adaptation of the $\mathcal{HC}$-Search framework to the problem of multi-label learning.

The MLS approach first defines a generic combinatorial search space over all possible multi-label outputs. Next, a search procedure (e.g. breadth-first or greedy search) is specified, which traverses the output space with the goal of uncovering high-quality outputs for a given input x. Importantly, for this search to be effective, it will often be necessary to guide it using a learned heuristic. Finally a learned cost function is used to score the set of outputs uncovered by the search procedure, and the least-cost one is returned. The effectiveness of the MLS approach depends on: 1) the ability of the search to uncover good outputs for given inputs, which for difficult problems will depend on the quality of the search heuristic $\mathcal{H}$, and 2) the ability of the cost function $\mathcal{C}$ to select the best of those outputs. We employ existing learning approaches proposed within the $\mathcal{HC}$-Search framework for learning effective heuristics and cost functions for these purposes as they are shown to be very effective in practice.

Our second contribution is to conduct a broad evaluation of several existing multi-label learning algorithms along with our MLS approach on a variety of benchmarks by employing diverse task loss functions. Our results demonstrate that the performance of existing algorithms tends to be very similar in most cases, and that our MLS approach is comparable and often better than all the other algorithms across different loss functions. Our results also identify particular ways where our approach can be improved.

5.1 Related Work

Typical approaches to multi-label learning decompose the problem into a series of independent binary classification problems, and employ a thresholding or ranking scheme to make predictions (Elisseeff and Weston, 2001; Read et al., 2011; Fürnkranz et al., 2008; Tsoumakas et al., 2010). The Binary Relevance (BR) method ignores correlations between output labels and learns one independent classifier for every label (Tsoumakas et al., 2010). The Classifier Chain (CC) (Read et al., 2011) approach learns one classifier for every label based on input x and the assignments to previous labels in a fixed ordering over labels. CC leverages the interdependencies between output labels to some extent, but it suffers from two major problems: 1) It is hard to determine a good ordering of the labels in the chain, 2) Errors can propagate from earlier predictions to later

[1] In a concurrent work to ours, the Condensed Filter Tree (CFT) algorithm was proposed for training loss-sensitive multi-label classifiers (Li and Lin, 2014).

ones (Ross and Bagnell, 2010; Hal Daumé III et al., 2009; Ross et al., 2011). To address some of these issues, researchers have employed ensembles of chains (ECC), beam search (Kumar et al., 2013) and monte carlo search (MCC) (Read et al., 2013) techniques to find a good ordering and for predicting the labels. All these label decomposition approaches try to optimize the Hamming loss.

Output coding methods try to exploit the correlations between output labels by coding them using a different set of latent classes. There are several output coding techniques for multi-label learning, including coding based on compressed sensing (Hsu et al., 2009), Principal Component Analysis (PCA) (Tai and Lin, 2012), and Canonical Correlation Analysis (CCA) (Zhang and Schneider, 2011). These methods either try to find a discriminative set of codes ignoring predictability, or vice versa. The recent max-margin output coding method (Zhang and Schneider, 2012) tries to overcome some of the drawbacks of previous coding approaches by searching for a set of codes that are both discriminative and predictable via a max-margin formulation. However, their formulation poses the problem of finding these codes as an intractable optimization problem for which they propose an approximate solution. These output coding approaches optimize a fixed, but unknown loss function.

Graphical modeling approaches including Conditional Random Fields (CML) (Ghamrawi and McCallum, 2005), Bayesian Networks (LEAD) (Zhang and Zhang, 2010), and Conditional Dependency Networks (CDN) (Guo and Gu, 2011) try to capture label dependencies, but unfortunately suffer from the intractability of the exact inference problem due to the high tree-width graphical structure. It is possible to employ approximate inference methods (e.g., Loopy Belief Propagation and MCMC), with the associated risk of converging to local optima or not converging at all. These methods try to optimize the structural log loss or some variant of it.

Probablistic Classifier Chains (PCC) (Dembczynski et al., 2010) estimate the conditional probablity of every possible label set for an input instance, and employ a Bayes optimal inference rule to optimize the given task loss function. However, the PCC framework suffers from two problems: 1) It is hard to accurately estimate the conditional probabilities, and 2) It is non-trivial to come up with the inference rule for a new loss function. Exact inference rules are known for a few loss functions: Hamming loss, Rank loss, and F1 loss (Dembczynski et al., 2010, 2011, 2012a, 2013). Approximate inference methods can be employed to optimize the Exact-Match loss (Dembczynski et al., 2012b; Kumar et al., 2013; Read et al., 2013).

The Structured Support Vector Machines (SSVMs) (Tsochantaridis et al., 2005) framework allows varying loss functions, but requires a loss-augmented inference routine for the given task

loss function, which is non-trivial if the loss function is non-decomposable. Existing multi-label prediction approaches based on this framework either resort to approximate inference or some form of convex relaxation for non-decomposable losses (Hariharan et al., 2010; Petterson and Caetano, 2010, 2011).

Label powerset (LP) methods reduce the multi-label learning problem to a multi-class classification problem, and optimize the Exact-Match loss. These approaches are very inefficient for training and testing. RAndom K-labELsets (RAKEL) is a representative approach of LP methods (Tsoumakas and Vlahavas, 2007). Some recent work has proposed a variant of RAKEL to optimize weighted Hamming loss (Lo et al., 2011). ML-kNN (Zhang and Zhou, 2007) is an extension of the traditional k-Nearest Neighbor classification algorithm for multi-label prediction. It is very expensive to make predictions with ML-kNN for large-scale training data.

5.2 Multi-Label Search Framework

In this section, we first describe the formal problem setup. Next, we give an overview of the $\mathcal{HC}$-Search framework followed by our instantiation for multi-label prediction problems and then describe the learning algorithms.

5.2.1 Problem Setup

A multi-label prediction problem specifies a space of inputs $\mathcal{X}$, where each input $x \in \mathcal{X}$ can be represented by a d dimensional feature vector; a space of outputs $\mathcal{Y}$, where each output $y = (y_1, y_2, \cdots, y_T) \in \mathcal{Y}$ is a binary vector of length T; 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 inputs to outputs whose predicted outputs have low expected loss with respect to the distribution $\mathcal{D}$.

5.2.2 Overview of the $\mathcal{HC}$-Search Framework

Recall that the $\mathcal{HC}$-Search framework for structured prediction is based on search in the output space $\mathcal{Y}$ and is parameterized by the following elements: 1) a search space $\mathcal{S}_o$, where each state

in the search space consists of an input-output pair (x, y) where y represents the potential output for the input x, 2) a time-bounded search strategy $\mathcal{A}$ (e.g., depth-limited greedy search), 3) a learned heuristic function $\mathcal{H} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$ in cases where the search strategy requires one, and 4) a learned cost function $\mathcal{C} : \mathcal{X} \times \mathcal{Y} \mapsto \Re$.

Given all of these elements and an input x, a prediction is made by first running the search procedure $\mathcal{A}$ (guided by $\mathcal{H}$ when appropriate), for a specified time bound τ. During the search a set of states is traversed, where each state represents a potential output for x. The cost function is employed to score each such output and the least-cost output is returned as the predicted label for x. The effectiveness of this approach depends on the quality of the search space (i.e., expected depth at which the target outputs can be located), the ability of the search procedure and heuristic function to guide the search to uncover high-quality outputs, and the quality of the cost function in terms of correctly scoring those outputs.

5.2.3 Multi-Label Search (MLS)

To instantiate the $\mathcal{HC}$-Search framework for multi-label prediction, we need to specify effective search spaces and search strategies that are appropriate for different multi-label prediction problems.

Search Space. The states of our multi-label search space correspond to input-output pairs (x, y), representing the possibility of predicting y as the multi-label output for x. In general, such a search space is defined in terms of two functions: 1) An *initial state function* I that takes an input x and returns an initial search state (x, y), and 2) A *successor function* S that takes a state as input and returns a set of child states. Given an input x, the search always begins at state $I(x)$ and then traverses the space by following paths allowed by the successor function.

In this work, we employ a simple search space for multi-label problems, which we call the *Flipbit-null* space. In particular, the initial state function I is defined as $I(x) = (x, \text{null})$, where null is the zero vector indicating that no labels are present. The successor function $S((x, y))$ returns all states of the form (x, y'), where y' differs from y in exactly one label position, i.e., the hamming distance between y and y' is 1. Thus, individual search steps in this space can be viewed as picking a particular output label and flipping its value. This space is effectively the search space underlying Gibbs sampling. Clearly the search space is complete, since for any input x it is possible to reach any possible output starting from the initial state.

Search Space Quality. The quality of a search space can be understood in terms of the ex-

pected amount of search needed to uncover the correct output y^*, which often increases monotonically with the expected depth of the target in the search space. In particular, for a given input-output pair (x, y^*), the target depth $d(x, y^*)$ is defined as the minimum depth at which we can find a state corresponding to the target output y^*. Clearly according to this definition, the expected target depth of the Flipbit-null space is equal to the expected number of non-zero labels. That is, for the Flipbit-null space we have,

$$\begin{aligned} \mathbf{d} &= \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ d\left(x, y^*\right) \\ &= \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ \left|y^*\right|_1 \end{aligned} \tag{5.1}$$

Thus, the expected target depth of the Flipbit-null space is related to the average sparseness of the label vectors. We observe that for several standard benchmarks the outputs are very sparse[2] (80 perecent of the benchmarks have sparsity less than 4), which makes the above search space very effective. To the best of our knowledge, we are not aware of any multi-label approach that *explicitly* exploits the sparsity property of multi-label problems.

Other Search Space Choices. One possible way to decrease the expected target depth, if necessary, would be to define more sophisticated search spaces that are tuned for particular types of multi-label problems. As a simple example, if the number of zero entries in the outputs is small, then it would be more effective to define the initial state of the Flipbit space to be the vector of all ones. The expected target depth would then be the expected number of zero output labels. Another way to reduce the expected target depth would be to use an existing multi-label approach P (e.g., Binary Relevance) to produce the initial state. The resulting Flipbit space can then be viewed as biasing the search toward this solution (e.g., see (Lam et al., 2013)). In this case, the expected target depth of the search space would be equal to the expected Hamming error of P on the multi-label problem. Finally, even more sophisticated spaces such as the Limited Discrepancy Search space (Doppa et al., 2012) defined in terms of a greedy classifier chain or a variant of its sparse version (Doppa et al., 2014b) could be employed.

Search Strategies. Recall that in our MLS approach, the role of the search procedure is to uncover high-quality outputs. We can consider depth-bounded breadth-first search (BFS), but unfortunately BFS will not be practical for a large depth k and/or a large number of output labels T. Even when BFS is practical, it generates a large number of outputs (T^k) that will make the cost function learning problem harder. Therefore, in this paper, we consider depth-limited

[2] http://mulan.sourceforge.net/datasets.html

greedy search guided by a (learned) heuristic function $\mathcal{H}$ as our search strategy. Given an input x, greedy search traverses a path of user specified length k through the search space, at each point selecting the successor state that looks best according to the heuristic. In particular, if s_i is the state at search step i, greedy search selects $s_{i+1} = \text{argmin}_{s \in S(s_i)} \mathcal{H}(s)$, where $s_0 = I(x)$. The time complexity of generating this sequence is $O(k \cdot T)$, which makes it much more practical than BFS for larger values of k. The effectiveness of greedy search is determined by how well $\mathcal{H}$ guides the search toward generating state sequences that contain high quality outputs. It is possible to consider other heuristic search strategies, such as best-first search and beam-search. However, in our experience so far, greedy search has proven sufficient.

5.2.4 Learning Algorithms

We first describe the loss decomposition of the $\mathcal{HC}$-Search approach along with its staged learning. Next, we briefly talk about the heuristic and cost function learning algorithms in the context of greedy search. In this work, we focus on learning linear $\mathcal{H}$ and $\mathcal{C}$ of the form $\mathcal{H}(x, y) = w_{\mathcal{H}} \cdot \Phi_{\mathcal{H}}$(x,y) and $\mathcal{C}(x, y) = w_{\mathcal{C}} \cdot \Phi_{\mathcal{C}}$(x,y), where $\Phi_{\mathcal{H}}$ and $\Phi_{\mathcal{C}}$ are feature functions that compute the feature vectors for $\mathcal{H}$ and $\mathcal{C}$ respectively, and $w_{\mathcal{H}}$ and $w_{\mathcal{C}}$ stand for their weights that will be learned from the training data.

Loss Decomposition and Staged Learning. As discussed in Chapter 4, for any heuristic function $\mathcal{H}$ and cost function $\mathcal{C}$, the overall loss of the $\mathcal{HC}$-Search approach $\mathcal{E}(\mathcal{H}, \mathcal{C})$ can be decomposed into the loss due to $\mathcal{H}$ not being able to generate the target output (generation loss $\epsilon_{\mathcal{H}}$), and the additional loss due to $\mathcal{C}$ not being able to score the best outputs generated by $\mathcal{H}$ correctly (selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$) . The loss decomposition can be mathematically expressed as follows:

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

where $y^*_{\mathcal{H}}$ is the best output that is generated by the search guided by $\mathcal{H}$ and $\hat{y}$ is the predicted output. The $\mathcal{HC}$-Search approach performs a staged-learning by first learning a heuristic $\mathcal{H}$ to minimize the generation loss $\epsilon_{\mathcal{H}}$, and then the cost function $\mathcal{C}$ is learned by minimizing the selection loss $\epsilon_{\mathcal{C}|\mathcal{H}}$, given the learned $\mathcal{H}$.

Heuristic Learning. The heuristic function $\mathcal{H}$ is trained via imitation learning. For a given training time bound τ_{max} and task loss function L, we perform greedy search with the loss function used as an oracle heuristic on every training example (x, y^*) (ties are broken randomly) and generate training data for imitation (see Algorithm 5). The imitation example R_t at each search step t consists of one ranking example for every candidate state $s \in S(s_{t-1}) \setminus s_t$ such that $\mathcal{H}(x, y_t) < \mathcal{H}(x, y)$, where (x, y_t) and (x, y) correspond to the input-output pairs associated with states s_t and s respectively. The aggregate set of imitation examples collected over all the training data is then given to a rank learner (e.g., Perceptron or SVM-Rank) to learn the parameters of $\mathcal{H}$.

Cost Function Learning. The cost function $\mathcal{C}$ is trained via cross-validation to avoid overfitting (see Chapter 4 for details). We divide the training data into k folds and learn k different heuristics, where each heuristic function $\mathcal{H}_i$ is learned using the data from all the folds excluding the i^{th} fold. We generate ranking examples for cost function learning using each heuristic function $\mathcal{H}_i$ on the data it was not trained on. Specifically, we perform greedy search guided by $\mathcal{H}_i$ to generate a set of outputs $\mathcal{Y}_{\mathcal{H}_i}(x)$, and generate ranking examples for any pair of outputs $(y_{best}, y) \in \mathcal{Y}_{best} \times \mathcal{Y}_{\mathcal{H}_i}(x) \setminus \mathcal{Y}_{best}$ such that $\mathcal{C}(x, y_{best}) < \mathcal{C}(x, y)$, where $\mathcal{Y}_{best}$ is the set of all best loss outputs. We give the aggregate set of ranking examples to a rank learner to learn the cost function $\mathcal{C}$.

Algorithm 5 Heuristic Function Learning for Greedy Search

Input: $\mathcal{D}$ = Training data, (I, S) = Search space, L = Loss function, τ_{max} = no. of training steps

Initialize the set of ranking examples $\mathcal{R} = \emptyset$
for each training example $(x, y^*) \in \mathcal{D}$ **do**
  $s_0 \leftarrow I(x)$ *// initial state*
  for each search step $t = 1$ to τ_{max} **do**
    Generate example R_t to imitate this search step
    Aggregate training data: $\mathcal{R} = \mathcal{R} \cup R_t$
    $s_t \leftarrow \text{argmin}_{s \in S(s_{t-1})} L(s)$ *// oracle search*
  end for
end for
$\mathcal{H}$ = Rank-Learner($\mathcal{R}$)
return heuristic function $\mathcal{H}$

5.3 Empirical Results

In this section, we evaluate our MLS approach along with several existing multi-label algorithms on a variety of benchmarks and evaluation measures.

5.3.1 Datasets

We employ nine benchmark[3] datasets for our evaluation. We selected these datasets based on the diversity of domains (text, images, audio and bio-informatics) and their popularity within the multi-label learning community. Table 5.2 presents the properties of different datasets. Ten percent of the training data were used to tune the hyper-parameters.

5.3.2 Experimental Setup

Evaluation Measures. We consider four diverse loss functions: Hamming loss, Exact-Match (0/1) loss, F1 loss, and Accuracy loss. F1 and Accuracy losses do not decompose over individual labels. They are defined as follows: F1 loss = 1 - $\frac{2\|\hat{y} \cap y^*\|_1}{\|\hat{y}\|_1 + \|y^*\|_1}$; Accuracy loss = 1 - $\frac{\|\hat{y} \cap y^*\|_1}{\|\hat{y} \cup y^*\|_1}$, where $\hat{y}$ is the predicted output and y^* is the correct output. When both the predicted labels and ground-truth labels are zero vectors, then we consider the loss to be *zero* for both F1 and Accuracy unlike existing software packages including Mulan and Meka[4] that consider the loss to be *one* in this case.

MLS Approach. We employ the simple Flipbit-null space and greedy search as described in Section 5.2. We use unary and pair-wise potential features for our heuristic and cost function representation, i.e., these functions are represented as $w \cdot \Phi(x, y)$, where w is the parameter vector to be learned, and $\Phi(x, y)$ is a feature vector that contains indicator features for the activation levels of all label pairs and features that are the cross product of the label space and input features composing x. We estimate the expected target depth $\mathbb{E}[d]$ of the Flipbit-null space for each dataset and use $2 * \lceil \mathbb{E}[d] \rceil$ steps for training and testing the heuristic function noting that we didn't see any improvements with larger timebounds. For cost function learning, we experimented with 3 folds and 5 folds, but larger folds didn't help much. We employ SVM-Rank (Joachims, 2006) as our base rank learner for both heuristic and cost function learning. The C parameter was

[3] http://mulan.sourceforge.net/datasets.html
[4] http://meka.sourceforge.net/

ALGORITHMS	Scene	Emotions	Medical	Genbase	Yeast	Enron	LLog	Slashdot	Tmc2007
a. Hamming Accuracy Results									
BR	86.90	77.10	98.50	99.80	78.30	94.00	97.70	94.50	**94.70**
CC	88.70	76.80	98.60	99.90	78.50	95.10	98.40	94.50	94.60
ECC	89.00	78.40	98.30	99.90	78.50	94.30	98.40	94.70	**94.70**
M2CC	89.80	78.50	98.30	99.90	78.10	94.50	98.50	94.90	94.60
CLR	89.10	78.40	97.90	96.60	77.00	94.00	97.10	92.70	94.50
CDN	89.40	80.30	98.40	99.60	78.10	94.70	97.70	94.60	94.60
CCA	88.59	79.05	97.79	99.19	79.05	93.66	95.10	94.60	94.22
PIR	87.81	67.99	98.79	**99.93**	77.18	94.66	97.05	94.11	94.34
SML	86.32	78.64	**98.83**	99.04	78.64	94.80	**99.60**	95.28	94.01
RML	88.11	79.04	**98.83**	99.89	79.71	95.25	98.46	95.48	94.68
DecL	90.12	81.89	98.79	99.80	79.67	95.40	98.40	95.22	93.91
MLS	**90.41**	**82.75**	**98.83**	99.80	**80.72**	**95.60**	98.50	**95.63**	94.10
b. Instance-based F1 Results									
BR	52.60	60.20	63.90	98.70	63.20	53.90	36.00	46.20	71.80
CC	59.10	57.50	64.00	99.40	63.20	53.30	26.50	44.90	70.30
ECC	68.00	62.60	65.30	99.40	64.60	59.10	32.20	50.20	72.70
M2CC	68.20	63.20	65.40	99.40	64.90	59.10	32.30	50.30	72.80
CLR	62.20	**66.30**	66.20	70.70	63.80	56.50	22.70	46.60	70.80
CDN	63.20	61.40	68.90	97.80	64.00	58.50	36.60	53.10	71.30
CCA	66.43	63.27	49.60	98.60	61.64	53.83	25.80	48.00	69.53
PIR	74.45	60.92	80.17	99.41	**65.47**	61.14	38.95	57.55	**73.73**
SML	68.50	64.32	68.34	**99.62**	64.32	57.46	34.95	55.73	71.63
RML	74.17	64.83	**80.73**	98.80	63.18	57.79	35.97	51.30	71.34
DecL	73.76	65.29	78.02	97.89	63.46	61.19	37.52	54.67	69.08
MLS	**75.89**	66.17	78.19	98.12	63.78	**62.34**	**39.76**	**57.98**	69.17
c. Instance-based Accuracy Results									
BR	48.50	52.30	61.50	98.00	52.30	44.10	27.80	41.90	62.30
CC	55.90	49.70	61.00	99.10	51.80	43.00	25.30	42.00	61.70
ECC	63.40	54.80	62.20	99.10	53.70	47.00	29.40	45.70	63.50
M2CC	63.70	55.00	62.90	99.10	53.40	47.10	29.50	46.00	63.70
CLR	62.50	56.80	58.10	56.10	51.30	42.70	17.20	38.10	60.00
CDN	61.50	56.80	64.70	96.60	52.80	47.00	32.30	48.40	62.10
CCA	62.12	55.40	60.10	98.20	50.82	42.90	19.60	43.30	62.38
PIR	67.87	49.75	**76.33**	**99.16**	**53.92**	49.16	34.42	**52.87**	63.76
SML	63.65	52.38	64.03	98.42	52.38	48.08	33.49	43.92	**63.77**
RML	67.23	53.91	75.90	98.17	52.41	47.98	33.16	47.27	63.05
DecL	66.19	54.17	74.23	97.91	50.45	49.87	35.78	48.77	58.56
MLS	**69.12**	**57.89**	74.98	98.34	51.23	**51.21**	**36.67**	**52.85**	59.76
d. Exact-Match Results									
BR	45.90	24.80	46.20	95.50	15.60	10.90	21.90	31.50	32.20
CC	47.50	25.20	47.80	98.00	19.20	12.50	22.60	32.00	**34.00**
ECC	52.00	28.20	43.60	98.00	19.60	11.90	22.40	32.50	33.50
M2CC	**59.63**	**32.20**	43.90	98.00	**21.50**	13.50	**24.70**	33.20	**34.00**
CLR	58.30	28.70	33.60	11.10	5.80	2.80	2.70	13.90	25.10
CDN	57.60	**32.20**	52.20	94.00	17.00	12.60	22.40	34.10	32.40
CCA	**59.63**	30.20	22.48	97.99	20.39	**15.70**	15.07	32.00	31.04
PIR	50.08	19.80	**64.65**	**98.49**	14.29	13.64	23.63	**38.80**	30.73
SML	52.84	30.06	62.17	91.51	15.05	12.15	24.23	35.03	31.75
RML	47.32	25.74	62.94	91.45	13.63	12.43	24.48	35.09	30.44
DecL	59.00	31.89	63.76	96.81	15.12	12.11	20.81	37.89	29.98
MLS	58.10	31.18	63.46	96.75	14.30	12.71	19.12	38.13	28.29

Table 5.1: Performance of different multi-label prediction algorithms.

Dataset	Domain	#TR	#TS	#F	#L	$\mathbb{E}[d]$
Scene	image	1211	1196	294	6	1.07
Emotions	music	391	202	72	6	1.86
Medical	text	333	645	1449	45	1.24
Genbase	biology	463	199	1185	27	1.25
Yeast	biology	1500	917	103	14	4.23
Enron	text	1123	579	1001	53	3.37
LLog	text	876	584	1004	75	1.18
Slashdot	text	2269	1513	1079	22	1.18
Tmc2007	text	21519	7077	500	22	2.15

Table 5.2: Characteristics of the datasets: the number of training (#TR) and testing (#TS) examples; number of features (#F); number of labels (#L); and the expected target depth of our Flipbit-null space ($\mathbb{E}[d]$).

tuned using the validation set. The MLS approach cannot work for Exact-Match loss[5], so we present the Exact-Match results by training with Hamming loss. In all other cases, we train for the given task loss function. Our base rank learner did not scale to the Tmc2007 dataset, so we performed our training on a subset of 5000 training examples.

Baseline Methods. Our baselines include Binary Relevance `BR` (Tsoumakas et al., 2010); Classifier Chain with greedy inference `CC` (Read et al., 2011); Ensemble of Classifier Chains `ECC` (Read et al., 2011); Monte Carlo optimization of Classifier Chains `M2CC` (Read et al., 2013); Calibrated Label Ranking `CLR` (Fürnkranz et al., 2008); Conditional Dependency Networks `CDN` (Guo and Gu, 2011); Canonical Correlation Analysis `CCA` (Zhang and Schneider, 2011); Plug-in-Rule approach `PIR` (Dembczynski et al., 2013); Submodular Multi-Label prediction `SML` (Petterson and Caetano, 2011); Reverse Multi-Label prediction `RML` (Petterson and Caetano, 2010); and Decomposed Learning `DecL` (Samdani and Roth, 2012). The last method, DecL, is a variant of our MLS approach that employs a different cost function learning algorithm by trying to rank the correct output y^* higher than all the outputs with a hamming distance of at most k from y^*. We employed the Meka package to run BR, CC, ECC, M2CC, CLR, and CDN. We ran the code provided by the authors for CCA[6], PIR[7], SML and RML[8].

For the methods that require a base classifier, we employed logistic regression with L2 regularization. The regularization parameter was tuned via 5-fold cross validation. We employed the natural ordering of labels for CC and 20 random orderings for ECC. We used 10 iterations

[5] We cannot differentiate between the outputs with loss *one*.

[6] http://www.cs.cmu.edu/∼yizhang1/files/AISTAT2011_Code.zip

[7] https://github.com/multi-label-classification/PCC

[8] http://users.cecs.anu.edu.au/∼jpetterson/

for learning the ordering, and 100 iterations for inference with M2CC. The parameters of CCA were tuned as described in the original paper. For PIR, we tuned the hyper-parameter λ via 5-fold cross-validation and report the results with their exact algorithm (EFP). The hyper-parameters for RML (λ), and SML (λ, C) were tuned based on the validation set. For DecL, we employed the largest value of k that was practical for training the cost function.

5.3.3 Results

Table 5.1 shows the accuracy results (higher is better) of different multi-label approaches with different evaluation measures. We can make several interesting observations from these results. First, the performance of several algorithms tend to be very similar in most cases. Second, our MLS approach performs comparably and often better than all other algorithms for all evaluation measures other than the Exact-Match accuracy, which MLS cannot optimize. The results of MLS for Tmc2007 are competetive with other methods even though MLS was trained only on one-fourth of the training data. Third, ECC performs better than CC as one would expect, and M2CC significantly improves over both CC and ECC showing the benefit of learning the ordering and performing a more elaborate search instead of greedy search. Fourth, PIR performs comparably or better than all other methods on F1 accuracy across all the datasets excluding Emotions. This behavior is expected because PIR is designed to optimize F1 loss.

We do not provide the error decomposition results for our MLS approach, but we would like to mention that the generation error for most datasets is close to zero, which means that most of our error is coming from the cost function, i.e., even though the heuristic is able to generate good outputs, cost function is not able to score them properly. Therefore, it would be productive to consider more powerful rank learners[9] (e.g., Regression trees (Mohan et al., 2011)) for cost function learning to improve the results.

5.4 Summary and Future Work

We introduced the Multi-Label Search (MLS) approach by adapting the $\mathcal{HC}$-Search framework for multi-label prediction problems. MLS can automatically adapt its training for a given task loss function, and can jointly predict all the labels without suffering from the intractability of inference. We show that the MLS approach gives comparable or better results than existing multi-

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

label approaches across several benchmarks and diverse loss functions. Future work should tackle the problem of designing an appropriate search space for the problem at hand, and the problem of learning in the context of more sophisticated search strategies when the expected depth of the search space is high and the greedy search is not effective.

Chapter 6: Conclusions and Future Work

In this dissertation, we studied a general framework for structured prediction called $\mathcal{HC}$-Search that integrates learning and search in a principled manner for solving structured prediction problems. Our framework *subsumes* existing frameworks for cost function learning and control knowledge learning. The $\mathcal{HC}$-Search framework involves *designing* a high-quality search space over structured outputs by leveraging the problem structure such that the target outputs (zero loss outputs) can be located at small depths; learning a heuristic function $\mathcal{H}$ to quickly *guide* the search towards high-quality outputs; and learning a cost function $\mathcal{C}$ to correctly *score* the outputs generated by the search procedure guided by heuristic $\mathcal{H}$.

We developed generic solutions for learning problems associated with all the key elements of this framework and an engineering methodology for applying this framework. We investigated several instantiations of the $\mathcal{HC}$-Search framework and showed that the $\mathcal{HC}$-Search framework achieves results in a wide range of structured prediction problems that significantly exceed the best previous results. Additonally, the error decomposition in terms of heuristic error (error due to not generating the optimal solution) and cost function error (error due to not selecting the best candidate solution generated by the heuristic) can be easily measured for a learned ($\mathcal{H}$,$\mathcal{C}$) pair and allow for an assessment of which function is more responsible for the overall error. The effectiveness of $\mathcal{HC}$-Search for a particular problem depends critically on the quality of the search space over structured outputs being employed, and our Limited Discrepancy Search (LDS) space played a major role in the current success of $\mathcal{HC}$-Search.

6.1 Lessons Learned

In this section, we describe the two most important lessons we learned from this work.

1. **$\mathcal{HC}$-Search is a "Divide and Conquer" solution approach with procedural knowledge injected into it.** Every component of the $\mathcal{HC}$-Search solution has a clearly pre-defined role, and contributes towards the overall goal by making the role of the other components easier. For example, the LDS space leverages greedy classifiers to reduce the target depth of the search space to make the heuristic learning problem easier; the heuristic function

$\mathcal{H}$ tries to make the cost function learning problem easier by generating high-quality outputs with as little (heuristically guided) search as possible; and it is sufficient for the cost function $\mathcal{C}$ to correctly score the best outputs generated by the heuristic.

2. **Inference in $\mathcal{HC}$-Search vs. Inference in CRF/SSVM.** Inference procedure in standard approaches such as CRF and SSVM involves scoring all possible outputs by the cost function. In contrast, in $\mathcal{HC}$-Search, the cost function needs to score only a small subset of candidate outputs generated by the search procedure guided by the heuristic $\mathcal{H}$, which is relatively easier.

6.2 Summary of Contributions

The main contribution of this dissertation is the "$\mathcal{HC}$-Search framework" for solving structured prediction problems; developing generic solutions for learning problems associated with all the key elements of this framework; and an engineering methodology for applying this framework to new problems. In particular, the contributions include the following.

- We adapted the basic idea of Limited Discrepancy Search (LDS) (Harvey and Ginsberg, 1995) to structured prediction by defining the *Limited Discrepancy Search Space* (Doppa et al., 2014b), a generic search space over outputs that leverages greedy classifers (Dietterich et al., 1995; Hal Daumé III et al., 2009), and related the quality of the LDS space to the quality of learned classifiers.

- We analyzed the computational complexity of learning within the $\mathcal{HC}$-Search framework. We identified 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. Guided by the decomposition, we developed a stage-wise approach to learning the heuristic and cost functions based on imitation learning.

- We evaluated the $\mathcal{HC}$-Search approach empirically on a number of benchmark problems arising in natural language processing and computer vision, compared it to state-of-the-art methods, and analyzed different dimensions of the framework. We showed that our framework yields significantly superior performance, and allows an informative error analysis and diagnostics.

- We developed a simple framework for multi-label prediction called Multi-Label Search (MLS) based on instantiating our $\mathcal{HC}$-Search framework to multi-label learning by *explicitly* exploiting the sparsity property of multi-label problems. We empirically evaluated our MLS framework along with many existing multi-label learning algorithms on a variety of benchmarks by employing diverse task loss functions. We showed that the performance of existing algorithms tends to be very similar in most cases, and that the MLS approach is comparable and often better than all the other algorithms across different loss functions.

6.3 Future Work

In this section, we list some important future directions for this line of research.

- The $\mathcal{HC}$-Search framework should be applied 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 et al. (2013), and multi-object tracking in complex sports videos Chen et al. (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.

- 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 using more expressive representations for the cost functions (e.g., boosted regression trees) 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).

- An important research problem is to investigate learning approaches to trade off speed and accuracy of inference in a principled manner. In $\mathcal{HC}$-Search, the most computationally demanding steps are the generation of candidate states. Hence, we can obtain speedups by generating fewer candidate states (e.g., via pruning). Therefore, future work should consider learning search space pruning rules to improve the efficiency of both training and making predictions.

- Another interesting line of work is to learn representations for heuristic and cost functions by trading off computational cost and value of information of features (Weiss and Taskar, 2013).

Bibliography

Shivani Agarwal and Dan Roth. Learnability of bipartite ranking functions. In *Proceedings of International Conference on Learning Theory (COLT)*, pages 16–31, 2005.

C. Andrieu, N. De Freitas, A. Doucet, and M. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 37(3):277–296, 1999.

Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data*. MIT Press, 2007. ISBN 0262026171.

Dhruv Batra, Payman Yadollahpour, Abner Guzmán-Rivera, and Gregory Shakhnarovich. Diverse m-best solutions in Markov random fields. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 1–16, 2012.

Justin A. Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research (JMLR)*, 1:77–112, 2000.

Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

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

Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. Structured learning with constrained conditional models. *Machine Learning Journal (MLJ)*, 88(3):399–431, 2012.

Chao Chen, Vladimir Kolmogorov, Yan Zhu, Dimitris Metaxas, and Christoph H. Lampert. Computing the M most probable modes of a graphical model. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

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

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

Michael Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of Association of Computational Linguistics (ACL) Conference*, 2002.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research (JMLR)*, 7:551–585, 2006.

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

Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 279–286, 2010.

Krzysztof Dembczynski, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. An exact algorithm for F-measure maximization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1404–1412, 2011.

Krzysztof Dembczynski, Wojciech Kotlowski, and Eyke Hüllermeier. Consistent multilabel ranking through univariate losses. In *Proceedings of International Conference on Machine Learning (ICML)*, 2012a.

Krzysztof Dembczynski, Willem Waegeman, and Eyke Hüllermeier. An analysis of chaining in multi-label classification. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 294–299, 2012b.

Krzysztof Dembczynski, Arkadiusz Jachnik, Wojciech Kotlowski, Willem Waegeman, and Eyke Hüllermeier. Optimizing the F-measure in multi-label classification: Plug-in rule approach versus structured loss minimization. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1130–1138, 2013.

Thomas G. Dietterich, Hermann Hild, and Ghulum Bakiri. A comparison of ID3 and backpropagation for English text-to-speech mapping. *Machine Learning Journal (MLJ)*, 18(1):51–80, 1995.

Justin Domke. Structured learning via logistic regression. In *Advances in Neural Information Processing Systems (NIPS)*, pages 647–655, 2013.

Janardhan Rao Doppa, Jun Yu, Prasad Tadepalli, and Lise Getoor. Chance-constrained programs for link prediction. In *Proceedings of NIPS Workshop on Analyzing Networks and Learning with Graphs*, 2009.

Janardhan Rao Doppa, Jun Yu, Prasad Tadepalli, and Lise Getoor. Learning algorithms for link prediction based on chance constraints. In *Proceedings of European Conference on Machine Learning (ECML)*, pages 344–360, 2010.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Output space search for structured prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2012.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: Learning heuristics and cost functions for structured prediction. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research (JAIR)*, 50: 369–407, 2014a.

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

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

André Elisseeff and Jason Weston. A kernel method for multi-label classification. In *Advances in Neural Information Processing Systems (NIPS)*, pages 681–687, 2001.

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

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

Alan Fern, Sung Wook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 25:75–118, 2006.

Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153, 2008.

Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*, pages 195–200, 2005.

Yoav Goldberg and Michael Elhadad. 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)*, pages 742–750, 2010.

Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1300–1305, 2011.

Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2005.

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

Bharath Hariharan, Lihi Zelnik-Manor, S. V. N. Vishwanathan, and Manik Varma. Large scale max-margin multi-label classification with priors. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 423–430, 2010.

William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 607–615, 1995.

Tamir Hazan and Raquel Urtasun. Efficient learning of structured predictors in general graphical models. *CoRR*, abs/1210.2346, 2012.

Klaus-Uwe Hoffgen, Hans-Ulrich Simon, and Kevin S. Van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1):114–125, 1995.

Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *Advances in Neural Information Processing Systems (NIPS)*, pages 772–780, 2009.

Liang Huang, Suphan Fayong, and Yang Guo. 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)*, pages 142–151, 2012.

Jiarong Jiang, Adam Teichert, Hal Daumé III, and Jason Eisner. Learned prioritization for trading off accuracy and speed. In *Advances in Neural Information Processing (NIPS)*, 2012.

T. Joachims. Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 217–226, 2006.

Matti Kääriäinen. Lower bounds for reductions. In *Atomic Learning Workshop*, 2006.

Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, and Dan Chazan. Phoneme alignment based on discriminative learning. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, pages 2961–2964, 2005.

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

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3):123–286, 2012.

Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Beam search algorithms for multilabel learning. *Machine Learning*, 92(1):65–89, 2013.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

Michael Lam, Janardhan Rao Doppa, Xu Hu, Sinisa Todorovic, Thomas Dietterich, Abigail Reft, and Marymegan Daly. Learning to detect basal tubules of nematocysts in SEM images. In *Proceedings of ICCV Workshop on Computer Vision for Accelerated Biosciences (CVAB)*. IEEE, 2013.

Chun-Liang Li and Hsuan-Tien Lin. Condensed filter tree for cost-sensitive multi-label classification. In *Proceedings of International Conference on Machine Learning (ICML)*, 2014.

Qi Li, Heng Ji, and Liang Huang. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 73–82, 2013.

Hung-Yi Lo, Ju-Chiang Wang, Hsin-Min Wang, and Shou-De Lin. Cost-sensitive multi-label learning for audio tag annotation and retrieval. *IEEE Transactions on Multimedia*, 13(3): 518–529, 2011.

Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Fern, Thomas G. Dietterich, and Prasad Tadepalli. Prune-and-score: Learning for greedy coreference resolution. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

David A. McAllester, Tamir Hazan, and Joseph Keshet. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1594–1602, 2010.

Ofer Meshi, David Sontag, Tommi Jaakkola, and Amir Globerson. Learning efficiently with approximate inference via dual losses. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 783–790, 2010.

Ananth Mohan, Zheng Chen, and Kilian Q. Weinberger. Web-search ranking with initialized gradient boosted regression trees. *Journal of Machine Learning Research - Proceedings Track*, 14:77–89, 2011.

Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy-belief propagation for approximate inference: An empirical study. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

Dennis Park and Deva Ramanan. N-best maximal decoders for part models. In *Proccedings of IEEE International Conference on Computer Vision (ICCV)*, pages 2627–2634, 2011.

Nadia Payet and Sinisa Todorovic. SLEDGE: Sequential labeling of image edges for boundary detection. *International Journal of Computer Vision (IJCV)*, 104(1):15–37, 2013.

James Petterson and Tibério S. Caetano. Reverse multi-label learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1912–1920, 2010.

James Petterson and Tibério S. Caetano. Submodular multi-label learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1512–1520, 2011.

Xian Qian, Xiaoqian Jiang, Qi Zhang, Xuanjing Huang, and Lide Wu. Sparse higher order conditional random fields for improved sequence labeling. In *Proceedings of International Conference on Machine Learning (ICML)*, 2009.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

Jesse Read, Luca Martino, and David Luengo. Efficient Monte Carlo optimization for multi-label classifier chains. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 3457–3461, 2013.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. *Journal of Machine Learning Research - Proceedings Track*, 9:661–668, 2010.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research - Proceedings Track*, 15:627–635, 2011.

Dan Roth and Wen tau Yih. Integer linear programming inference for conditional random fields. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 736–743, 2005.

Rajhans Samdani and Dan Roth. Efficient decomposed learning for structured prediction. In *Proceedings of International Conference on Machine Learning (ICML)*, 2012.

Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

David Sontag, Ofer Meshi, Tommi Jaakkola, and Amir Globerson. More data means less inference: A pseudo-max approach to structured learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2181–2189, 2010.

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

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. 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)*, pages 725–733, 2011.

Charles A. Sutton and Andrew McCallum. Piecewise training for structured prediction. *Machine Learning Journal (MLJ)*, 77(2-3):165–194, 2009.

Umar Syed and Rob Schapire. A reduction from apprenticeship learning to classification. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2253–2261, 2010.

Farbound Tai and Hsuan-Tien Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

Benjamin Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of International Conference on Machine Learning (ICML)*, 2004.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, 2005.

Grigorios Tsoumakas and Ioannis P. Vlahavas. Random k -labelsets: An ensemble method for multilabel classification. In *Proceedings of European Conference on Machine Learning (ECML)*, pages 406–417, 2007.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. 2010.

Grigorios Tsoumakas, Min-Ling Zhang, and Zhi-Hua Zhou. Introduction to the special issue on learning from multi-label data. *Machine Learning*, 88(1-2):1–4, 2012.

Julia Vogel and Bernt Schiele. Semantic modeling of natural scenes for content-based image retrieval. *International Journal of Computer Vision (IJCV)*, 72(2):133–157, 2007.

David Weiss. Structured prediction cascades code. `http://code.google.com/p/structured-cascades/`, 2014.

David Weiss and Benjamin Taskar. Structured prediction cascades. *Journal of Machine Learning Research - Proceedings Track*, 9:916–923, 2010.

David Weiss, Ben Sapp, and Ben Taskar. Sidestepping intractable inference with structured ensemble cascades. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2415–2423, 2010.

David J. Weiss and Ben Taskar. Learning adaptive value of information for structured prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 953–961, 2013.

Michael L. Wick, Khashayar Rohanimanesh, Sameer Singh, and Andrew McCallum. Training factor graphs with reinforcement learning for efficient MAP inference. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2044–2052, 2009.

Michael L. Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. Samplerank: Training factor graphs with atomic gradients. In *Proceedings of International Conference on Machine Learning (ICML)*, 2011.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research (JMLR)*, 10:1571–1610, 2009.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Iterative learning of weighted rule sets for greedy search. In *Proceedings of International Conference on Automated Planning and Systems (ICAPS)*, pages 201–208, 2010.

Nan Ye, Wee Sun Lee, Hai Leong Chieu, and Dan Wu. Conditional random fields with high-order features for sequence labeling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2196–2204, 2009.

Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. Max-violation perceptron and forced decoding for scalable MT training. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1112–1123, 2013.

Min-Ling Zhang and Kun Zhang. Multi-label learning by exploiting label dependency. In *Profeedings of International Conference on Knowledge Discovery in Databased (KDD)*, pages 999–1008, 2010.

Min-Ling Zhang and Zhi-Hua Zhou. ML-kNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120, 1995.

Yi Zhang and Jeff G. Schneider. Multi-label output codes using canonical correlation analysis. *Journal of Machine Learning Research - Proceedings Track*, 15:873–882, 2011.

Yi Zhang and Jeff G. Schneider. Maximum margin output coding. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1223–1230, 2012.