Randomized Greedy Search for Structured Prediction: Amortized Inference and Learning

Chao Ma[1†] , **F A Rezaur Rahman Chowdhury**[2†] , **Aryan Deshwal**[2] ,
Md Rakibul Islam[2] , **Janardhan Rao Doppa**[2] and **Dan Roth**[3]
[1]School of EECS, Oregon State University
[2]School of EECS, Washington State University
[3]Department of Computer and Information Science, University of Pennsylvania
machao@oregonstate.edu, {f.chowdhury, aryan.deshwal, mdrakibul.islam, jana.doppa}@wsu.edu,
danroth@seas.upenn.edu

Abstract

In a structured prediction problem, we need to learn a predictor that can produce a structured output given a structured input (e.g., part-of-speech tagging). The key learning and inference challenge is due to the exponential size of the structured output space. This paper makes four contributions towards the goal of a computationally-efficient inference and training approach for structured prediction that allows to employ complex models and to optimize for non-decomposable loss functions. First, we define a simple class of randomized greedy search (RGS) based inference procedures that leverage classification algorithms for simple outputs. Second, we develop a RGS specific learning approach for amortized inference that can quickly produce high-quality outputs for a given set of structured inputs. Third, we plug our amortized RGS inference solver inside the inner loop of parameter-learning algorithms (e.g., structured SVM) to improve the speed of training. Fourth, we perform extensive experiments on diverse structured prediction tasks. Results show that our proposed approach is competitive or better than many state-of-the-art approaches in spite of its simplicity.

1 Introduction

Structured prediction (SP) tasks arise in several domains including NLP, computer vision, and computational biology. In a structured prediction problem, the goal is to learn a mapping from structured inputs to structured outputs. For example, in semantic labeling of images, the structured input is an image and the structured output is a labeling of the image regions. The main learning and inference challenge in structured prediction is due to the size of the structured output space, which is exponential in the size of the input.

A standard approach to structured prediction is to learn a scoring function $F(x, y)$ to score a candidate structured output y given a structured input x [Lafferty *et al.*, 2001; Tsochantaridis *et al.*, 2004]. Given such a scoring function and a new input x, the output computation then involves finding the maximum scoring output (aka *Argmax inference* problem). Unfortunately, exactly solving the argmax problem is often intractable and efficient solutions exist only for simple representations over input-output pairs (e.g., small factor sizes). However, many real-world tasks require complex representations for making accurate predictions, which significantly increase the time-complexity of inference. Since inference algorithms are used in the inner loop of structured learning to induce weights of $F(x, y)$, it increases the training time to perform learning on large-scale data. Also, there is no general approach to perform learning to optimize *non-decomposable loss functions* (e.g., F1 loss): prior methods require decomposability to support "loss augmented inference".

This paper has two main motivations. First, there is little work on using advances in machine learning to successfully improve the speed of solving inference problems in structured prediction literature [Srikumar *et al.*, 2012; Kundu *et al.*, 2013; Chang *et al.*, 2015b; Namaki *et al.*, 2017; Pan and Srikumar, 2018]. Our goal is to bridge the general area of speedup learning [Fern, 2010] with structured prediction for potential cross-fertilization of ideas and new applications [Namaki *et al.*, 2017]. By viewing inference procedures as computational search processes will allow us to study generic approaches to address speedup learning problems arising in structured prediction. Second, randomized greedy search (RGS) based inference method referred as *RGS(0)* — starting solutions of RGS are selected randomly — is shown to be empirically successful for couple of NLP tasks [Zhang *et al.*, 2014; 2015] and theoretical results based on Gibbs generalization error characterize its efficacy [Honorio and Jaakkola, 2016]. However, there are two main weaknesses of RGS(0): a) RGS(0) will not perform well on SP tasks with large structured outputs (large number of output variables) and large number of candidate labels; and b) RGS(0) requires a large number of random restarts to uncover near-optimal structured outputs. This results in high inference time. Since inference is repeatedly used in the inner loop of SP algorithms for learning weights of scoring function, training time is also high. Indeed, our experiments demonstrate these weaknesses

† First two authors contributed equally.

of RGS(0). We address the above mentioned challenges using a simple, yet very effective approach based on RGS based inference. This paper makes the following contributions:

- Define a simple class of RGS based inference procedures referred as RGS(α) that leverages classification algorithms for simple outputs. The parameter $\alpha \in [0,1]$ determines the fraction of output variables assigned by a learned classifier to generate starting solutions.
- Develop a RGS specific learning approach for *amortized inference* that can quickly produce high-quality outputs for a given set of structured inputs. The key idea is to learn an evaluation function to select good starting solutions to improve the accuracy of search.
- Employ amortized RGS inference inside the inner loop of parameter-learning algorithms (e.g., structured SVM) to improve the speed of training.
- Perform comprehensive experiments on ten diverse SP tasks including sequence labeling, multi-label classification, co-reference resolution, and image segmentation. Results show that our approach is competitive or better than many state-of-the-art approaches in spite of its simplicity. The code and data is publicly available on github: https://github.com/nkg114mc/rgs-struct

2 Problem Setup

A structured prediction problem specifies a space of structured inputs $\mathcal{X}$, a space of structured outputs $\mathcal{Y}$, and a non-negative *loss function* $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re^+$ such that $L(x, y', y^*)$ is the loss associated with labeling a particular input x by output y' when the true output is y^*. We are provided with a training set of input-output pairs $\{(x, y^*)\}$ drawn from an unknown target distribution $\mathcal{D}$. The goal is to return a function from structured inputs to outputs whose predictions have low expected loss w.r.t distribution $\mathcal{D}$.

Without loss of generality, let each structured output $y \in \mathcal{Y}$ be represented as d discrete variables $v_1, v_2, \cdots, v_d$ and each variable v_i can take candidate values from a set $C(v_i)$. For part-of-speech (POS) tagging, v_i stands for POS tag of a word and $C(v_i)$ is the list of all POS tags. Since our algorithms will be learning functions over input-output pairs, as is standard in structured prediction, we assume the availability of a *feature function* $\Phi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^m$ that computes an m dimensional feature vector for any pair. We assume a linear scoring function $F(x, y) = w \cdot \Phi(x, y)$ to score a candidate input-output pair, where $w \in \Re^m$ stands for *weights/parameters*. Our goal is to learn w such that for each training example (x, y^*), the score of the correct output is higher than score of any other candidate output y, i.e., $F(x, y^*) > F(x, y)$.

3 RGS(α) Inference Procedure

We define a simple class of RGS inference procedures referred as *RGS(α)* that leverage IID classifiers.

Inference Problem. Given a structured input x and a parameterized function $F(x, y)$ to score candidate outputs, the problem of finding the highest scoring candidate output is called *"Argmax" inference problem*: $\hat{y} = \arg\max_y F(x, y)$. We need to solve inference problems during both training and testing. Therefore, we need an efficient and accurate solver.

Algorithm 1 RGS(α) Inference Solver

Input: x: structured input, $F(x, y)$: scoring function, h: learned IID classifier, $1 - \alpha \in [0, 1]$: randomness in starting output, R_{max}: maximum number of restarts

```
1: Initialization: $y_{best} \leftarrow$ random output
2: for each restart $r = 1$ to $R_{max}$ do
3:     $y_{start} \leftarrow$ assign $\alpha$ fraction of output variables using classifier
       $h$ and remaining variables randomly
4:     $(\text{TRAJ}_r, y_{end}) \leftarrow \texttt{Greedy}((x, y_{start}), \mathbb{S}, F(x, y))$
5:     if $F(x, y_{end}) > F(x, y_{best})$ then
6:         $y_{best} \leftarrow y_{end}$ // Update the best scoring output
7:     end if
8: end for
9: return $y_{best}$, the predicted structured output
```

RGS(α) Inference Solver. We employ randomized greedy search (RGS) based solvers that perform search in complete structured output spaces, where each search state s is of the form (x, y). Given a starting state (x, y_{start}), greedy search traverses a path through the search space, at each point selecting the best successor state according to the scoring function $F(x, y)$ until reaching a local optima (x, y_{end}). We denote the greedy search trajectory by TRAJ. RGS performs greedy search from multiple starting states and selects the best local optima y_{best}. The time-complexity of RGS inference is $\mathcal{O}(R \cdot d \cdot k \cdot T)$, where R is the number of restarts, d is the number of output variables, k is the maximum number of candidate values for output variables, and T is the average number of search steps to reach local optima. The quality of predicted output y_{best} depends critically on the starting states and the number of restarts. RGS(α) is a variant of RGS whose starting states are selected using a parameter $\alpha \in [0, 1]$ and a learned IID classifier h. Given a structured input x, we select a starting state output as follows: assign α fraction of output variables using classifier h and remaining variables uniformly at randomly. RGS(0) is the instantiation employed in prior work [Zhang *et al.*, 2014].

IID Classifier h. It is a multi-class classifier that varies from one SP task to another. The classifier h predicts the label for each node / output variable in the structured output *independently*, i.e., ignores the label dependencies based on the node features (unary potentials). For example, in sequence labeling, h predicts each output token independently based on the features of input token. We employed off-the-shelf logistic regression implementation to learn the IID classifier from the structured input-output training examples $\{(x, y^*)\}$ using unary features in all our experiments.

Successor Function $\mathbb{S}$. Given a state $s = (x, y)$, we generate a successor state by changing the value of *exactly one* output variable in y with one of the other candidate values. Therefore, the number of successors is equal to $\sum_{i=1}^{d} (|C(v_i)| - 1)$.

Effect of α. The value of α controls the trade-off between exploitation and exploration. Specifically, the following two properties are relevant: 1) minimum depth at which target outputs y^* can be located if we fully expand the search tree rooted at the initial state via successor function $\mathbb{S}$ (referred as *target depth*); and 2) diversity of starting outputs. Consider the following two extreme cases. If $\alpha = 1$, the expected value of the target depth is proportional to the error of classifier h and is lowest. We can view this as biasing the search toward the output produced by classifier h resulting in no diversity in the starting states. If $\alpha = 0$, the expected target depth value is proportional to the error of a random classifier and is highest. In this case, diversity in starting states is very high. For other values of α between 0 and 1, the resulting trade-off lies in between these two extremes. α is a hyper-parameter. The best value of α depends on hardness and scale of the structured prediction task. For example, it may be beneficial to have non-zero α value when the no. of output variables d is large and our experiments corroborate this hypothesis. In our experiments, we select the best α based on validation data.

Advantages of RGS Inference. Some of the main advantages of search-based inference methods including RGS are as follows: a) They can easily handle global constraints to perform inference over valid outputs. We just need to prune all successor states that violate the global constraints. b) We can employ any higher-order features as part of $\Phi(x, y)$ with negligible computational overhead. c) Allows us to optimize non-decomposable loss functions (e.g., F1 loss) due to the naturally ability of RGS to solve loss-augmented inference problem with arbitrary loss function L: $\hat{y} = \arg\max_y F(x, y) + L(x, y, y^*)$. Indeed, we demonstrate these advantages of RGS(α) inference in our experiments.

4 Learning for Amortized RGS Inference

In this section, we describe a learning approach to improve the speed of solving inference problems using RGS inference solver (aka amortized RGS inference).

Motivation. We need to solve inference problems for multiple inputs during both training and testing. The naive approach is to run inference solver *independently* on each input example. It is conceivable that we can learn useful knowledge while solving inference problems on past input examples to improve the speed of inference on future examples [Fern, 2010; Srikumar *et al.*, 2012]. For example, we can improve the speed of RGS(α) inference solver if we can select better starting states. Therefore, we learn evaluation functions that can estimate the promise of a candidate output as a starting point for greedy search [Boyan and Moore, 2001].

Amortized RGS Inference Problem. Given a set of structured inputs $D_x = \{x^i\}$ and a scoring function $F(x, y)$ to score candidate outputs, the goal is to reduce the time-complexity of finding accurate structured outputs for all inputs in D_x using the RGS(α) inference solver [Srikumar *et al.*, 2012].

Learning Approach. The key idea is to learn an evaluation function $E(x, y) = \theta \cdot \Phi(x, y)$ from past searches to select good starting states for greedy search [Boyan and Moore, 2001]. Suppose (x, y_{start}) is a candidate starting state and (x, y_{end}) is the local optima state obtained by performing greedy search guided by $F(x, y)$. We want to learn the parameters θ such that $E(x, y_{start}) = F(x, y_{end})$. If we can learn an accurate $E(x, y)$, then we can select good starting states using E.

Algorithm 2 Amortized RGS Inference

```
Input: $D_x = \{x^i\}_{i=1}^n$: structured inputs, $F(x,y)$: learned scoring function, $h$: IID classifier, $1-\alpha \in [0,1]$: randomness in starting output, $\theta$: old weights of evaluation function $E$
 1: for each input example $x^i \in D_x$ do
 2:     $y^i_{start} \leftarrow$ assign $\alpha$ fraction of output variables using classifier $h$ and remaining variables randomly
 3:     $\hat{y}^i \leftarrow y^i_{start}$
 4: end for
 5: repeat
 6:     Initialization: set of regression examples $\mathcal{R} \leftarrow \emptyset$
 7:     for each input example $x^i \in D_x$ do
 8:         // Greedy search guided by $F$
 9:         $(\text{TRAJ}^i, y^i_{end}) \leftarrow \texttt{Greedy}\big((x, y^i_{start}), F(x,y)\big)$
10:         if $F(x, y^i_{end}) > F(x, \hat{y}^i)$ then
11:             $\hat{y}^i \leftarrow y^i_{end}$ // Update the best scoring output
12:         end if
13:     end for
14:     // Generate training data to improve $E$
15:     for each output $y$ on $\text{TRAJ}^i$ do
16:         if $E(x^i, y) \neq F(x^i, y^i_{end})$ then
17:             Add the following regression example to $\mathcal{R}$: $\Phi(x^i, y)$ as input and $F(x^i, y^i_{end})$ as output
18:         end if
19:     end for
20:     // Improve $E$ using new training data
21:     $E = \text{ONLINE-REGRESSION-LEARNER}(\mathcal{R}, \theta)$
22:     for each input example $x^i \in D_x$ do
23:         // Search guided by $E$ for finding good starting solutions
24:         $(\text{TRAJ}^i, y^i_{reset}) \leftarrow \texttt{Greedy}\big((x^i, y^i_{end}), E(x^i, y)\big)$
25:         if $y^i_{reset} = y^i_{end}$ then
26:             $y^i_{start} \leftarrow$ assign $\alpha$ fraction of output variables using classifier $h$ and remaining variables randomly
27:         else
28:             $y^i_{start} \leftarrow y^i_{reset}$
29:         end if
30:     end for
31: until convergence
32: return $D_y$, the set with $\hat{y}^i$ for each input $x^i \in D_x$ and $\theta$, the new weights of evaluation function $E$
```

Our iterative learning approach for amortized RGS inference is shown in Algorithm 2 and illustrated in Fig 1. We initialize the starting output y^i_{start} for each input $x^i \in D$. We repeat the following four main steps until convergence (predicted outputs don't change in two consecutive iterations). Before each iteration, we reset the set of regression examples: $\mathcal{R} = \emptyset$. **Step 1:** For each input $x^i \in D_x$, we perform greedy search from the corresponding y^i_{start} to get local optima output y^i_{end}. Subsequently, the best scoring output $\hat{y}^i$ is updated for each input in D. (lines 7-13) **Step 2:** Collect regression examples to improve E from greedy search trajectories for all inputs in D. (lines 15-19) **Step 3:** Update the parameters of evaluation function E by passing regression examples $\mathcal{R}$ to an online regression learner. **Step 4:** Select good starting state for each input $x^i \in D_x$ using the updated E (lines 22-

30). We perform greedy search guided by E from y^i_{end} to get the local optima output y^i_{reset}. If y^i_{end} and y^i_{reset} are same, picking y^i_{reset} as a starting solution won't help in improving $\hat{y}^i$ and we pick a new starting solution as done in RGS(α). Otherwise, we pick y^i_{reset} as the next starting state.

Online Regression Learner. We employ a simple online learner based on gradient descent to learn E: learning rate $\eta = 0.1$ and five online learning iterations. More sophisticated approaches including the AdaGrad algorithm can also be used for this purpose.

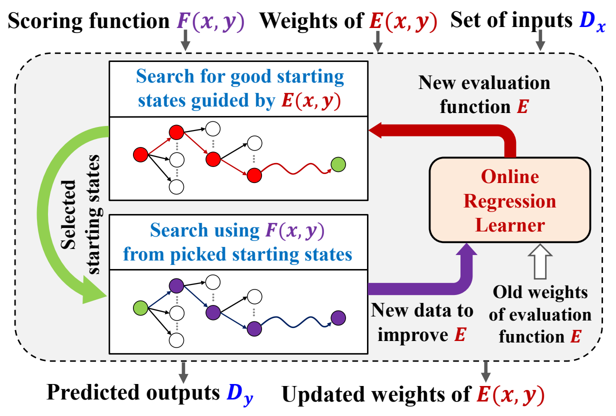
Scoring function $F(x,y)$
Weights of $E(x,y)$
Set of inputs D_x
Search for good starting states guided by $E(x,y)$
New evaluation function E
Online Regression Learner
Selected starting states
Search using $F(x,y)$ from picked starting states
New data to improve E
Old weights of evaluation function E
Predicted outputs D_y
Updated weights of $E(x,y)$

Figure 1: High-level overview of amortized RGS inference.

Connection to Imitation Learning. We can view the above learning approach as an instance of *imitation learning* (IL) [Daumé *et al.*, 2009; Ross *et al.*, 2011]. In traditional IL, expert demonstrations are provided as training data (e.g., demonstrations of a human expert driving a car) In our setting, the greedy search trajectories from step 1 correspond to expert demonstrations and evaluation function E correspond to the learned policy. We can view steps 2 and 3 as an instantiation of the IL algorithm DAgger [Ross *et al.*, 2011] with a small change. We do not aggregate data from all the previous iterations to induce policy via supervised learning as time is a first-order citizen in the amortized inference process. Therefore, we perform online updates to the policy parameters incrementally based on the new stream of feedback data from each iteration. The convergence theory for DAgger algorithm only depends on the use of a no-regret learning algorithm such as follow-the-regularized-leader (FTRL) [Ross *et al.*, 2011]. So convergence results will hold by using an online no-regret learning algorithm.

Usage of Amortized RGS Inference. The amortized RGS inference approach can be employed in two scenarios:

1) After learning $F(x,y)$**.** In this case, $F(x,y)$ is fixed and parameters of $E(x,y)$ get updated as needed while processing new inputs at the test-time. We initialize the parameters of $E(x,y)$ before testing by calling Algorithm 2 with all structured input-output training examples.

2) During learning $F(x,y)$**.** In this case, $F(x,y)$ keeps changing over different structured learning iterations and the parameters of $E(x,y)$ get updated as needed while performing inference with respect to the current $F(x,y)$.

Remark 1. We learn the evaluation function on training examples conditioned on the scoring function $F(x,y)$ learned via structured learning. Subsequently, we update the weights of $E(x,y)$ as needed while processing inputs at the test-time.

5 Structured Learning with Amortized RGS

Standard approaches for structured learning (SL) including CRFs, structured SVMs, and structured perceptron repeatedly call the inference solver on one or more training inputs to learn the weights of the scoring function $F(x,y)$. Therefore, we can employ amortized RGS (`A-RGS`) inference solver to improve the training speed. To optimize a given loss function L, we need to solve the loss-augmented inference problem for each training input x: $\hat{y} = \arg\max_y F(x,y) + L(x,y)$. We can employ amortized RGS by passing $F(x,y)+L(x,y)$ to score outputs instead of $F(x,y)$. Algorithm 3 shows a generic template for structured learning that can be instantiated to get existing algorithms. By varying $|D'|$, we can get online ($|D'| = 1$), mini-batch ($|D'| < |D|$), and full batch ($|D'| = |D|$) training methods. In this work, we experiment with structured SVM training using dual co-ordinate descent (DCD) optimization approach noting that any other SL approach can be used. We have two hyper-parameters for learning the weights of $F(x,y)$ with A-RGS inference solver: C for structured SVM and α for A-RGS(α).

Algorithm 3 Structured Learning with Amortized RGS

Input: $D = \{x, y^*\}$, structured input-output training examples, L: loss function, h: learned IID classifier, $1-\alpha \in [0,1]$: randomness in starting output

```
1: Initialize the weights of scoring function $F$ and evaluation function $E$: $w \leftarrow 0$ and $\theta \leftarrow 0$
2: repeat
3:     Select a batch $D' \subseteq D$ for loss-augmented inference
4:     // Call Amortized-RGS inference solver
5:     $(D'_y, \theta) \leftarrow \texttt{A-RGS}(D', F(x,y) + L(x,y), h, 1-\alpha, \theta)$
6:     // Training to improve $F$ using $D'_y$
7:     Update weights $w$ using (aggregate) data from $D'_y$
8: until convergence
9: return $w$, weights of the learned scoring function $F$
```

RGS/A-RGS falls under the category of under-generating inference methods (i.e., explores a subset of the structured output space) [Finley and Joachims, 2008]. Therefore, Theorem 3 from [Chang *et al.*, 2015b] is also applicable for our case with the DCD optimization algorithm. In short, it only requires exact inference every τ successive inference calls to learn the correct weights for $F(x,y)$.

6 Experiments and Results

Structured Prediction Tasks and Datasets. We evaluate our approach on diverse tasks including sequence labeling, multi-label classification, coreference resolution, and image segmentation. We employ five sequence labeling datasets. **1) Handwriting Recognition:** We consider two variants [Daumé *et al.*, 2009]: one fold for training and remaining nine folds for testing in `HW-Small`, and vice-versa in `HW-Large`. **2) NETtalk Stress:** The task is to assign one of the 5 stress labels to each letter of a word. **3) NETtalk Phoneme:** Similar to stress task except the goal is to assign one of the 51 phoneme labels. The training/testing split of NETtalk is 1000/1000. **4) Protein:** The aim is to predict secondary structure of amino-acid residues. There training/testing split is 111/17. **5) Twitter POS tagging:** 25 POS labels dataset consisting of 1000-tweet OCT27TRAIN, 327-tweet OCT27DEV, 547-tweet DAILY547 as test set [Tu and Gimpel, 2018]. We employ three multi-label datasets, where the goal is to predict a binary vector corresponding to the relevant labels. **6) Yeast:** There are 14 labels and training/testing split of 1500/917. **7) Bibtex:** There are 159 labels and training/testing split of 4800/2515. **8) Bookmarks:** There are 208 labels and training/testing split of 60000/27856. We employ one coreference resolution dataset, where the goal is to cluster a set of textual mentions. **9) ACE2005:** This is a corpus of English documents with 50 to 300 gold mentions in each document. We follow the standard training/testing split of 338/117 [Durrett and Klein, 2014]. We employ one image segmentation dataset, where the goal is to label each pixel in an image with its semantic label. **10) MSRC:** This dataset contains 591 images and 21 labels. We employ standard training/testing split of 276/256, and each image was pre-segmented into around 700 patches with SLIC algorithm.

Experimental Setup. We employ `Illinois-SL` library for learning the weights of F via structured SVM approach. We employ a validation set to tune the hyper-parameters: C for Structured SVM and $\alpha \in [0, 1]$ for RGS inference. For MSRC and ACE2005, we use the standard development set and employ 20 percent of training data as validation set for other datasets. For image segmentation, we employed the unary features from [Lucchi *et al.*, 2011]. For coreference resolution, we extended the `Berkeley` system [Durrett and Klein, 2014] by formulating coreference resolution as producing a left-linking tree [Ma *et al.*, 2014]. We only perform experiments with gold mentions. We consider features $\Phi(x, y)$ consisting of unary and pairwise features (first-order) for both F and E for the simplest configuration. All experiments were run on a machine with dual processor 6 Core 2.67Ghz Intel Xeon CPU and 48GB memory. To account for randomness, we repeat each experiment 10 times, and report the mean and variance of metrics.

Evaluation Metrics. For sequence labeling task, we employ Hamming accuracy. For multi-label classification task, we use three popular metrics: Hamming accuracy = $\frac{\|y \cap y^*\|_1}{|y|}$; Example-F1 = $\frac{2\|y \cap y^*\|_1}{\|y\|_1 + \|y^*\|_1}$; and Example accuracy = $\frac{\|y \cap y^*\|_1}{\|y \cup y^*\|_1}$, where y and y^* are the predicted and ground truth outputs respectively. For coreference resolution task, we employ standard metrics including MUC, B^3, CEAF$_e$, and CoNLL score. We also consider mention-wise Hamming accuracy, which is defined as the fraction of mentions with correct links. For image segmentation task, we measure the pixel-wise classification accuracy: a) *Class Average:* average over the accuracies for each class label; and b) *Global:* global Hamming accuracy regardless of the class labels.

Error Analysis of RGS and A-RGS Inference. We report two different accuracy metrics over testing examples for diagnostic purposes. Suppose $Y'(x)$ be the set of all candidate outputs uncovered by search for a given input x. We compute the predicted output $\hat{y}$ for these accuracy metrics as follows: **a) Prediction:** $\hat{y} = \arg\max_{y \in Y'(x)} F(x, y)$ (best scoring output uncovered by search); and **b) Generation:** $\hat{y} = \arg\min_{y \in Y'(x)} L(x, y, y^*)$ (best loss output uncovered by search). Note that generation accuracy is greater than or equal to prediction accuracy. Generation accuracy is associated with the effectiveness of RGS/A-RGS inference in uncovering high-quality candidate structured outputs.

6.1 Results of RGS Inference with $\alpha = 0$

In prior work [Zhang *et al.*, 2014; 2015], RGS(0) was tested on two NLP tasks minimally. So we robustly tested RGS(0) along different dimensions — number of restarts, complexity of joint features $\Phi(x, y)$, loss functions — and also performed error analysis. Due to space constraints, we present the detailed results in appendix. **Summary of results:** a) Accuracy improves with more number of restarts and saturates at 100 restarts. Variance over accuracy decreases with more restarts improving the stability of results; b) Accuracy improves with higher-order features at negligible computational overhead; and c) Can successfully optimize arbitrary loss functions, and best results are obtained when training and testing is done with the same loss function.

Table 1 shows the best results obtained with RGS(0) via 50 restarts and highest-order features: pairs, triples, and quadruples of labels for sequence labeling; pairs of labels for multi-label classification; features over entity and entity-mention pairs for coreference resolution; and unary and global features in the form of normalized histogram for number of super-pixels with different labels for image segmentation.

a. Sequence Labeling

	HW-Small	**HW-Large**	**Phoneme**	**Stress**	**TwtPos**	**Protein**
Cascades	89.18	97.84	82.59	80.49	-	-
HC-Search	89.96	97.79	85.71	83.68	-	-
CRF	80.03	86.89	78.91	78.52	-	62.44
SEARN	82.12	90.58	77.26	76.15	-	-
BiLSTM	83.18	92.50	77.98	76.55	88.8	61.26
BiLSTM-CRF	88.78	95.76	81.03	80.14	89.2	62.79
Seq2Seq(Beam=1)	83.38	93.65	78.82	79.62	89.1	62.90
Seq2Seq(Beam=5)	86.92	96.72	82.19	80.96	89.9	63.34
Seq2Seq(Beam=20)	89.38	98.95	82.31	81.50	90.2	63.81
RGS(0)	92.32	97.83	82.28	80.84	89.9	62.75
RGS(α)	92.56	97.96	82.45	81.00	90.2	65.20
BiLSTM-RGS	91.83	98.15	82.55	81.10	90.1	64.19

b. Multi-label Classification

	Yeast			**Bibtex**			**Bookmarks**		
	Hamm	ExmF1	ExAcc	Hamm	ExmF1	ExAcc	Hamm	ExmF1	ExAcc
MLS	80.72	63.78	51.23	-	-	-	-	-	-
SPEN(E2E)	79.6	63.8	52.0	98.5	42.1	36.8	99.1	35.6	29.3
DVN	78.9	63.8	51.9	98.5	44.7	37.2	99.1	37.1	30.1
InfNet	79.4	63.6	51.7	98.1	42.2	37.1	99.2	37.6	30.9
RGS(0)	80.04	63.90	52.18	98.12	44.11	36.65	99.13	36.88	31.46
RGS(α)	80.10	63.90	52.90	98.62	44.86	36.78	99.15	36.98	31.58

c. Coreference Resolution (ACE 2005)

	MUC	BCube	CEAF$_e$	CoNLL
Berkeley	81.41	74.7	72.93	76.35
RGS(0)	80.07	74.13	71.25	75.15
RGS(α)	82.18	76.57	74.01	77.58

d. Image Segmentation (MSRC)

	Global	Average
ICCV2011	85	77
CRF-CNN	91.1	90.5
RGS(0)	81.27	73.14
RGS(α)	85.29	78.92
RGS(α)-CNN	91.53	90.28

Table 1: Comparison with the state-of-the-art.

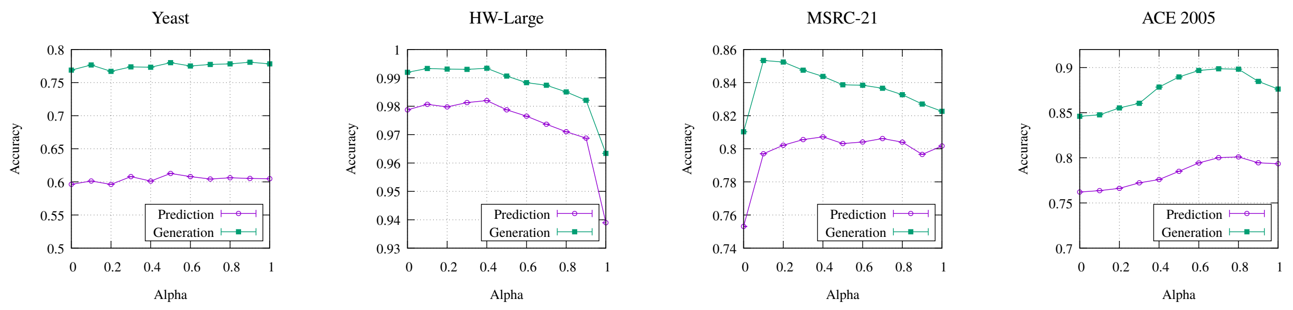

Figure 2: Prediction and generation accuracy of RGS(α) on development set for different values of α.

6.2 Results of RGS Inference with non-zero α

We present results of RGS with non-zero α values for training and testing. A non-zero α will provide a better starting output when compared to $\alpha = 0$. This can be potentially beneficial for harder tasks and/or when the number of output variables is very large. For this experiment, we employ 50 restarts, highest-order features, and optimize Hamming loss except for Yeast (F1 loss). We select the best α from $\{0, 0.1, 0.2, \cdots, 1.0\}$ via performance on the validation data.

RGS(0) vs. RGS with Best α

Table 1 shows the results for RGS with best α. Results are better with best α and accuracy improvement is higher for image segmentation and coreference resolutions, which are harder tasks with large number of output variables. Figure 2 shows the the effect of varying α on the validation data. Recall that α allows us to trade-off diversity (reflected via generation accuracy) and quality of the starting output for greedy search. For tasks with small structured outputs (Yeast and HW-Large), RGS(α) inference is likely to be more sensitive to the diversity of initial outputs as the main bottleneck is generation accuracy. However, the quality of the initial output is important for tasks with large structured outputs (MSRC and ACE2005). As α increases, generation accuracy starts decreasing after reaching the peak, but prediction accuracy does not decrease proportionally as improvement in the quality of starting output balances some of this negative effect.

Comparison to State-of-the-Art.

We compare the accuracy results of RGS with best α to state-of-the-art (SOTA) approaches including CRFs [Lafferty *et al.*, 2001], SEARN [Daumé *et al.*, 2009], CASCADES [Weiss and Taskar, 2010], HC-Search [Doppa *et al.*, 2014a], and recent deep learning approaches. Table 1 shows the prediction accuracies of different algorithms ('-' indicates we were not able to generate results for those cases).

For sequence labeling problems, we report the best published SOTA results [Doppa *et al.*, 2014a]. Performance of RGS(α) approach is comparable or better than SOTA for handwriting recognition and slightly lower than HC-Search for Stress and Phoneme datasets (HC-Search employs the high-quality limited discrepancy search space [Doppa *et al.*, 2014b]). We also performed experiments using BiLSTM, BiLSTM-CRF, and Seq2Seq with beam search[1] [Wiseman and Rush, 2016] as baselines. BiLSTM-CRF corresponds to using the learned unary features from Bi-LSTM and adding pairwise potentials to learn a CRF model. BiLSTM-RGS corresponds to using the learned unary features from Bi-LSTM with RGS(α). We report the accuracy with different decoding beam sizes in $\{1, 5, 20, 50\}$. The performance of Seq2Seq saturates at beam size 20 for all datasets. We find that RGS based approach consistently performs better or comparable to these baselines.

For multi-label classification, we compare to multi-label search (MLS) [Doppa *et al.*, 2014c], deep value networks (DVN) [Gygli *et al.*, 2017], end-to-end structured prediction energy networks (SPEN) [Belanger *et al.*, 2017], and SPEN with inference network (InfNet) [Tu and Gimpel, 2018]. RGS approach is competitive or slightly better when compared to DVN, SPEN(E2E) and SPEN(InfNet) (all of them learn a non-linear function).

For image segmentation, we compare to CRF-CNN [Liu *et al.*, 2015]. Performance of RGS approach with unary features from [Lucchi *et al.*, 2011] is lower than CRF-CNN. However, when we employ CNN based unary features (RGS(α)-CNN), performance is comparable to SOTA. For coreference resolution task, we compare to the Berkeley system [Durrett and Klein, 2014]. RGS approach performs better on all metrics.

6.3 Results of A-RGS(α) for Test-Time Inference

We present results comparing amortized RGS with baseline RGS for test-time inference. We employ the best α value from previous experiments for all the below results.

Time vs. Accuracy Curve. We perform this experiment on entire testing set (say $|D|$). The baseline RGS is run with 50 restarts. Figure 3 shows the time vs. accuracy curves. We plot the point corresponding to the cumulative time and accuracy of RGS over $|D|$ examples. For amortized RGS, we plot the anytime curve based on the results over different iterations. We make two observations. First, amortized RGS quickly uncovers high-quality outputs that are very close to final predictions from RGS. Second, given the same time-bound as RGS, accuracy of amortized RGS is same (HW-Large) or better (Yeast, ACE2005, MSRC) than RGS.

Fine-grained Analysis via Histogram of Required Restarts. To understand the speedup comparison between A-RGS and RGS, we plot the histogram of proportion of input examples as a function of the *required restarts* to

[1]Our seq2seq implementation is derived from https://github.com/JayParks/tf-seq2seq.

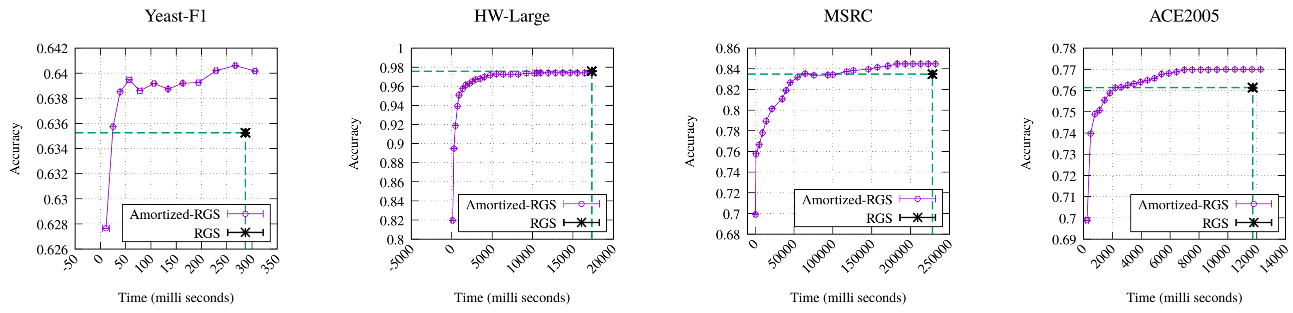

Figure 3: Time vs. accuracy curves for amortized RGS and RGS.

uncover the predicted output when they are run for a fixed number of restarts (set to 20 for this experiment).

Suppose for an input example x, the best scoring output according to $F(x, y)$ was found in k^{th} iteration, then we say that the *required restarts* of RGS/A-RGS for input x is k_x. Small values of k_x indicate faster inference (i.e., higher speedup). We compute the required restarts value k_x for each testing input x and plot the proportion of testing inputs for different candidate values of required restarts. Figure 4 shows the histograms comparing RGS and A-RGS. Results show that with A-RGS more input examples have relatively smaller required restarts value when compared to RGS. This phenomenon is pronounced and clear for both MSRC and ACE2005 (harder tasks). Table 2 shows the average value of required restarts over the entire testing set for different tasks. A-RGS has significantly lower value when compared to RGS resulting in large speedup improvements.

Name	**Yeast-F1**	**HW-Large**	**MSRC**	**ACE05**
RGS	2.050±0.016	2.182±0.301	9.343±0.869	5.359±0.713
A-RGS	1.824±0.017	1.874±0.398	5.175±0.834	3.239±0.382

Table 2: Average *required restarts* of RGS/A-RGS inference.

Inference Time. We measure the raw inference time with RGS/A-RGS over the entire testing set and compare with inference in DVN [Gygli *et al.*, 2017] and SPEN(E2E) [Belanger *et al.*, 2017] wherever possible. We employ the public implementations of DVN and SPEN(E2E), which were readily usable for the multi-label tasks. Table 3 shows the inference time results. A-RGS is 3 to 5 times faster than the baseline RGS approach. The speedup factor for A-RGS is higher for tasks with large structured outputs, where local optima challenge is significant and each iteration of greedy search is computationally expensive. A-RGS is better than DVN and SPEN(E2E) on Yeast, but SPEN(E2E) performs significantly better than both A-RGS and DVN on Bibtex and Bookmarks. We do not fully understand the reasons for this behavior and a thorough investigation is part of our immediate future work.

6.4 Results for Structured Learning with ARGS

We present results comparing structured learning (SL) with amortized RGS (Algorithm 3) and baseline RGS referred to as DCD(RGS) and DCD(A-RGS) in terms of training time and prediction accuracy. We keep the setup similar to above experiments. We train structured SVM with entire training set

Testing Time Results (milli seconds)

Name	**Yeast**	**Bibtex**	**Bookmarks**	**HWLarge**	**MSRC**	**ACE05**
SPEN(E2E)	1274	5791	63073	-	-	-
DVN	5504	18086	211448	-	-	-
RGS	444	69890	288058	17323	9451	282864
A-RGS	48	20925	98921	4812	2294	55355

Table 3: Inference time results (milli seconds).

Training Time Results (minutes)

Name	**Yeast**	**Bibtex**	**Bkmrks**	**HWLrg**	**MSRC**	**ACE05**
SPEN(E2E)	19	114	237	-	-	-
DVN	4	20	204	-	-	-
DCD(RGS)	9	95	392	71	115	171
DCD(A-RGS)	5	32	319	44	27	39

Table 4: Training time results (minutes).

as batch D', which is the standard configuration employed in practice. We define the training speedup factor of A-RGS as T_{RGS}/T_{ARGS}, where T_{ARGS} and T_{RGS} stand for training time using A-RGS and RGS inference solvers respectively. Results for raw training time, accuracy, and speedup factor of A-RGS when compared to RGS are shown in Table 5. We make two observations. First, training time with amortized RGS is significantly less when compared to training with RGS and accuracies are almost same. Second, speedup factor τ is large for harder tasks (ACE2005 and MSRC).

	DCD(RGS)		DCD(A-RGS)		Speedup
Datasets	Time(min.)	Acc.	Time(min.)	Acc.	factor
Yeast-F1	17±0.71	63.35±1.61E-6	10±0.83	63.20±2.97E-6	1.69±9.40E-3
HW-Large	91±1.03	97.79±9.01E-4	32±1.16	97.51±3.13E-4	2.82±1.61E-3
ACE2005	179±8.83	77.58±7.85E-4	44±6.35	77.32±1.26E-3	4.01±1.57E-1
MSRC	279±10.3	83.47±6.06E-3	89±8.51	84.02±1.84E-3	3.13±4.28E-2

Table 5: Results of training with DCD(RGS) and DCD(A-RGS).

We also compare the raw training time of SL via DCD optimization using RGS/A-RGS inference solvers — DCD(RGS) and DCD(A-RGS) — with both DVN and SPEN(E2E) whenever possible. Table 4 shows the training time results. DVN and SPEN(E2E) perform better on Bibtex and Bookmarks, whereas DCD(A-RGS) perform comparably or better on Yeast. We investigated the reasons for this behavior by profiling the DCD based structured learning algorithm with A-RGS. We found that on an average a large fraction (∼89%) of the training time was spent on updating the dual weights and only a small fraction (∼11%) on performing inference. We did not account for the pre-training time for SPEN in our report training time results. SPEN(E2E) employs a gra-

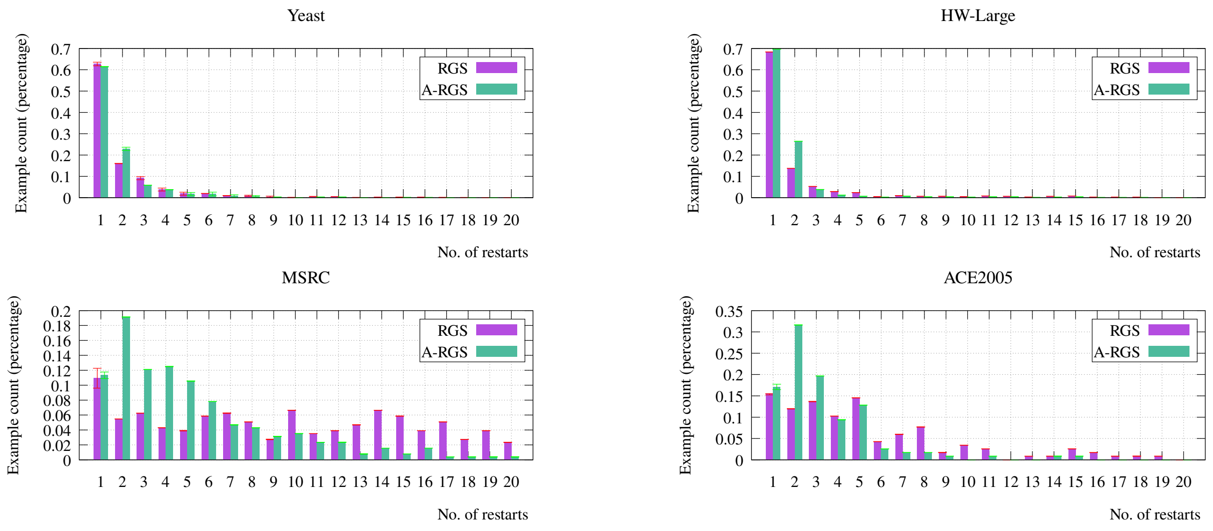

Figure 4: Histograms of percentage of examples for different values of *required restarts*.

dient based inference in the relaxed continuous space, whose strength is speed. However, it has two weaknesses: a) Can not handle global constraints to search over valid outputs; and b) Prone to overfitting. It may be a fruitful research direction to combine the best of both gradient-based inference and RGS inference methods.

7 Related Work

There are many approaches to solve structured prediction tasks with varying strengths and weaknesses. They include generalization of standard classification approaches such as CRFs [Lafferty *et al.*, 2001], structured SVM [Tsochantaridis *et al.*, 2004], structured perceptron [Collins, 2002], and their integration with deep learning (SPEN [Belanger *et al.*, 2017] and DVN [Gygli *et al.*, 2017]); search-based approaches that learn different forms of search control knowledge such as greedy policies (SEARN [Daumé *et al.*, 2009], DAgger [Ross *et al.*, 2011], LOLS [Chang *et al.*, 2015a], and variants [Xie *et al.*, 2015]), heuristic functions (LaSO and its variants [Daumé and Marcu, 2005; Xu *et al.*, 2009; Huang *et al.*, 2012]), heuristic and cost functions (HC-Search [Doppa *et al.*, 2014a; Lam *et al.*, 2015]), and coarse-to-fine knowledge (Cascades [Weiss and Taskar, 2010]).

Our work builds on the recent success of RGS based inference on couple of NLP tasks [Zhang *et al.*, 2014; 2015] and associated theory [Honorio and Jaakkola, 2016]. We studied a generalized RGS inference solver — RGS(α) — that leverages IID classifiers towards the goal of improving RGS. We also provided a learning approach to enable amortized RGS inference. Iterated conditional modes (ICM) inference algorithm [Pan and Srikumar, 2018] has some similarity to RGS. In ICM, in each local search step, only the candidate labels of a *single* output variable are considered, whereas labels of *all* output variables are considered in RGS. The principle behind our amortized RGS approach can be applied for ICM also.

Our amortized RGS inference method is very closely related to algorithms for amortized inference using integer linear programming (ILP) solvers [Srikumar *et al.*, 2012; Chang *et al.*, 2015b]. The key distinction is in the abstraction that allows for amortization. In our case, we view inference as a computational search process and learn generalized search control knowledge to improve the speed of reasoning. By adopting this viewpoint, we can treat ILP inference as a white box (i.e., branch and bound search) to consider alternate algorithms to achieve amortized inference [He *et al.*, 2014]. Our overall amortized inference based learning approach can be seen as an instantiation of HC-Search [Doppa *et al.*, 2014a] for randomized greedy search. Evaluation function E and scoring function F correspond to H and C respectively, but they are learned using different methods. Our work is also related to speedup learning (SL) [Fern, 2010] and can be seen as an instance of inter-problem SL. One of our primary motivations of this paper is to bridge the two areas of speedup learning and structured prediction for cross-fertilization of ideas. In fact, our amortized RGS approach builds on the ideas behind the STAGE algorithm from SL literature.

8 Summary and Future Work

We studied a randomized greedy search (RGS) based approach for structured prediction that leverages classifiers for simple outputs. We developed learning methods to improve the speed of making high-quality predictions using RGS. Experimental results show that in spite of its simplicity, our overall approach is very effective in terms of accuracy and speed of inference/training. Future work includes extending the RGS framework to multi-task structured prediction problems [Ma *et al.*, 2017], and to support learning in weak-supervision setting.

Acknowledgements

This work was supported in part by NSF grants #1910213 and #1845922, in part by Contracts W911NF15-1-0461 and HR0011-17-S-0048 with the US Defense Advanced Research Projects Agency (DARPA), and in part by grant W911NF-19-1-0162 from the US Army Research Office (ARO).

References

[Belanger *et al.*, 2017] David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *ICML*, 2017.

[Boyan and Moore, 2001] Justin Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *JMLR*, 2001.

[Chang *et al.*, 2015a] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

[Chang *et al.*, 2015b] Kai-Wei Chang, Shyam Upadhyay, Gourab Kundu, and Dan Roth. Structural learning with amortized inference. In *AAAI*, 2015.

[Collins, 2002] Michael Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *ACL*, 2002.

[Daumé and Marcu, 2005] Hal Daumé, III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

[Daumé *et al.*, 2009] Hal Daumé, III, John Langford, and Daniel Marcu. Search-based structured prediction. *MLJ*, 2009.

[Doppa *et al.*, 2014a] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A learning framework for search-based structured prediction. *JAIR*, 2014.

[Doppa *et al.*, 2014b] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *JMLR*, 2014.

[Doppa *et al.*, 2014c] Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. HC-Search for multi-label prediction: An empirical study. In *AAAI*, 2014.

[Durrett and Klein, 2014] Greg Durrett and Dan Klein. A joint model for entity analysis: Coreference, typing, and linking. In *TACL*, 2014.

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

[Finley and Joachims, 2008] Thomas Finley and Thorsten Joachims. Training structural SVMs when exact inference is intractable. In *ICML*, 2008.

[Gygli *et al.*, 2017] Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep value networks learn to evaluate and iteratively refine structured outputs. In *ICML*, 2017.

[He *et al.*, 2014] He He, Hal Daumé III, and Jason Eisner. Learning to search in branch and bound algorithms. In *NIPS*, 2014.

[Honorio and Jaakkola, 2016] Jean Honorio and Tommi S. Jaakkola. Structured prediction: From gaussian perturbations to linear-time principled algorithms. In *UAI*, 2016.

[Huang *et al.*, 2012] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *NACCL*, 2012.

[Kundu *et al.*, 2013] Gourab Kundu, Vivek Srikumar, and Dan Roth. Margin-based decomposed amortized inference. In *ACL*, 2013.

[Lafferty *et al.*, 2001] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[Lam *et al.*, 2015] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G. Dietterich. HC-search for structured prediction in computer vision. In *CVPR*, 2015.

[Liu *et al.*, 2015] Fayao Liu, Guosheng Lin, and Chunhua Shen. CRF learning with CNN features for image segmentation. *Pattern Recognition*, 2015.

[Lucchi *et al.*, 2011] Aurélien Lucchi, Yunpeng Li, Xavier Boix Bosch, Kevin Smith, and Pascal Fua. Are spatial and global constraints really necessary for segmentation. In *ICCV*, 2011.

[Ma *et al.*, 2014] Chao Ma, Janardhan Rao Doppa, John Walker Orr, Prashanth Mannem, Xiaoli Z. Fern, Thomas G. Dietterich, and Prasad Tadepalli. Prune-and-score: Learning for greedy coreference resolution. In *EMNLP*, 2014.

[Ma *et al.*, 2017] Chao Ma, Janardhan Rao Doppa, Prasad Tadepalli, Hamed Shahbazi, and Xiaoli Z. Fern. Multi-task structured prediction for entity analysis: Search-based learning algorithms. In *ACML*, 2017.

[Namaki *et al.*, 2017] Mohammad Hossein Namaki, F. A. Rezaur Rahman Chowdhury, Md Rakibul Islam, Janardhan Rao Doppa, and Yinghui Wu. Learning to speed up query planning in graph databases. In *ICAPS*, 2017.

[Pan and Srikumar, 2018] Xingyuan Pan and Vivek Srikumar. Learning to speed up structured output prediction. In *ICML*, 2018.

[Ross *et al.*, 2011] Stéphane Ross, Geoffery Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[Srikumar *et al.*, 2012] Vivek Srikumar, Gourab Kundu, and Dan Roth. On amortizing inference cost for structured prediction. In *EMNLP-CoNLL*, 2012.

[Tsochantaridis *et al.*, 2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[Tu and Gimpel, 2018] Lifu Tu and Kevin Gimpel. Learning approximate inference networks for structured prediction. In *ICLR*, 2018.

[Weiss and Taskar, 2010] David Weiss and Ben Taskar. Structured prediction cascades. In *AISTATS*, 2010.

[Wiseman and Rush, 2016] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*, 2016.

[Xie *et al.*, 2015] Jun Xie, Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Z. Fern, Thomas G. Dietterich, and Prasad Tadepalli. Learning greedy policies for the easy-first framework. In *AAAI*, 2015.

[Xu *et al.*, 2009] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning linear ranking functions for beam search with application to planning. *JMLR*, 10:1571–1610, 2009.

[Zhang *et al.*, 2014] Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. Greed is good if randomized: New inference for dependency parsing. In *EMNLP*, 2014.

[Zhang *et al.*, 2015] Yuan Zhang, Chengtao Li, Regina Barzilay, and Kareem Darwish. Randomized greedy inference for joint segmentation, POS tagging and dependency parsing. In *NAACL-HLT*, 2015.