Learning A* underestimates : Using inference to guide inference

Gregory Druck
University of Massachusetts
Amherst MA
gdruck@cs.umass.edu

Mukund Narasimhan
Live Labs, Microsoft Corp.
Redmond WA
mukundn@microsoft.com

Paul Viola
Live Labs, Microsoft Corp.
Redmond WA
viola@microsoft.com

Abstract

We present a technique for speeding up inference of structured variables using a priority-driven search algorithm rather than the more conventional dynamic programing. A priority-driven search algorithm is guaranteed to return the optimal answer if the priority function is an underestimate of the true cost function. We introduce the notion of a probable approximate underestimate, and show that it can be used to compute a probable approximate solution to the inference problem when used as a priority function. We show that we can learn probable approximate underestimate functions which have the functional form of simpler, easy to decode models. These models can be learned from unlabeled data by solving a linear/quadratic optimization problem. As a result, we get a priority function that can be computed quickly, and results in solutions that are (provably) almost optimal most of the time.

Using these ideas, discriminative classifiers such as semi-Markov CRFs and discriminative parsers can be sped up using a generalization of the A* algorithm. Further, this technique resolves one of the biggest obstacles to the use of A* as a general decoding procedure, namely that of coming up with a admissible priority function. Applying this technique results in a algorithm that is more than 3 times as fast as the Viterbi algorithm for decoding semi-Markov Conditional Markov Models.

1 Introduction

Dynamic programming is a widely used technique for decoding probabilistic models with structured outputs such as Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), semi-Markov CRFS, and Stochastic Context Free Grammars (SCFGs). While dynamic programming does yield a polynomial time algorithm for decoding these models, it may still be too slow (Charniak et al. 1998). For example, finding the optimal parse in a SCFG requires $O(n^3)$ time, where n is the number of tokens in the input. When SCFGs are used for decoding extremely large inputs (such as in information extraction applications (Viola et al. 2005) or bioinformatics applications (Durbin et al. 1999), an $O(n^3)$ algorithm may be excessively expensive. Even for simpler models like HMMs, for which decoding is $O(n)$, the hidden constants (a quadratic dependence on the number of states) can make dynamic programming unusable when there are many states. As a result, a number of alternatives to dynamic programming have been proposed such as Beam search (Ratnaparkhi 1999; Roark 2001; Collins 2004) best-first decoding (Charniak et al. 1998; Caraballo et al. 1997), thresholding and multiple-pass decoding (Goodman 1997; Weinberg 2006) and A* (Klein et al. 2003; Goel et al. 1999). Neither beam search nor best-first decoding are guaranteed to find the optimal solution. While A* is guaranteed to find the optimal solution, using it requires finding admissible underestimates (defined in Section 2).

Both A* and best-first search fall into the class of algorithms we call priority-based search techniques. A priority queue of partial solutions is maintained, and at each step, the partial solution with the lowest value of the priority function is taken off the queue. This partial solution is expanded to generate other partial/complete solutions which are then added to the queue. This process continues until a complete solution is taken off the priority queue, at which point the search stops. Best-first search uses the cost of the current solution as the priority function guiding its search, while A* uses the sum of the cost of the current solution and an optimistic estimate (underestimate) of the cost of completing the solution.

Since these priority-based schemes have to perform additional work at each step as compared to dynamic programming, the priority function has to prune away a substantial part of the search space in order to be effective. Further, since the priority function has to be computed for each step, for each partial solution, we need to able to compute the priority function quickly. Therefore, for A* to be effective (i.e., faster than Viterbi), the underestimate needs to be "tight", and fast to compute. This is one of the main obstacles to the general use of A*: coming up with tight inexpensive underestimates can be difficult.

So, we have algorithms which are fast, but offer no optimality guarantees (best-first decoding and beam search), and algorithms which prune the search space to find the optimal solution, but are often not as fast as the approximate algorithms. A further obstacle in using A* is that it is not always clear how to generate an admissible underestimate (especially in a discriminative setting), and hence A* cannot always be used.

Recently, a number of algorithms such as (Collins et. al 2004; Daumé et. al 2005) have been proposed to directly learn a model that can be decoded efficiently. These approaches tie the model and the decoding procedure, while the procedure presented in this paper allows the model to be learned independently of the decoding procedure. Given the model, a priority function is produced which, when used in a priority driven search procedure, is guaranteed to produce solutions which are close to optimal with high probability. For a given model, multiple priority functions can be generated, each representing a different tradeoff between approximation quality and computational requirement.

2 Notation and Preliminaries

The prediction/inference problem is to find a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space such that

$$f(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} \mathsf{cost}(\mathbf{y}|\mathbf{x})$$

When there is an underlying probabilistic model, the scoring function $\mathsf{cost}(\mathbf{y}|\mathbf{x})$ is typically the negative log likelihood $-\log p(\mathbf{y}|\mathbf{x})$. However, there are other models where the cost function is not derived from a purely probabilistic model (such as margin based models, voting based models, and loss based models).

In many problems, the input and output spaces have associated structure. For example, the space $\mathcal{X}$ might be the space of sequences of observations, (such as words or nucleic/amino acids), and the output space $\mathcal{Y}$ might be the space of sequences of labels (such as part-of-speech tags or coding/non-coding binary labels). In these problems, the size of the input/output domains is exponential in the length of the sequences, and hence exhaustive search cannot be used to find $\operatorname{\mathsf{argmin}}_{\mathbf{y} \in \mathcal{Y}} \mathsf{cost}(\mathbf{y}|\mathbf{x})$. In some cases the cost function also has associated structure, such as the Markov property, which allows for computing $\operatorname{\mathsf{argmin}}_{\mathbf{y} \in \mathcal{Y}} \mathsf{cost}(\mathbf{y}|\mathbf{x})$ in time polynomial in the length of the sequence.

We start with a concrete example, that of labeling sequences of observations where the input space is $\mathcal{X} = \mathcal{O}^n$, the output space is $\mathcal{Y} = \mathcal{L}^n$, and $\mathcal{O}$ and $\mathcal{L}$ are the set of observations and labels respectively. Note however, that the techniques presented in this paper apply equally to a number of other structured prediction problems.

Each element $\mathbf{l} = \langle l_1, l_2, \ldots, l_n \rangle \in \mathcal{L}^n = \mathcal{Y}$, represents an assignment of a label to every observation in the input. We will call $\mathbf{l}_{[\mathbf{1:k}]} = \langle l_1, l_2, \ldots, l_k \rangle$ a partial output; it is an assignment of labels to a prefix of the input sequence. We say that the cost function satisfies the Markov property if it assigns a cost to each partial output satisfying

$$\mathsf{cost}(\mathbf{l}_{[\mathbf{1:k+1}]}|\mathbf{x}) = \mathsf{cost}(\mathbf{l}_{[\mathbf{1:k}]}|\mathbf{x}) + \phi_k(l_k, l_{k+1}|\mathbf{x})$$

Once $\phi_k(l_k, l_{k+1}|\mathbf{x})$ and $\mathsf{cost}(\mathbf{l_1}|\mathbf{x})$ are specified, the value of $\mathsf{cost}(\mathbf{l}|\mathbf{x})$ can be computed for every label sequence $\mathbf{l} \in \mathcal{L}^n$.

We can formulate the search for the optimal solution as a shortest path problem in a graph $G = (V, E)$ constructed as follows. The node set V consists of all pairs $\{\langle t, l \rangle\}_{1 \leq t \leq n, l \in \mathcal{L}}$, where t is the time step of the node, and l is the label of the node. There are edges from node $\langle t, l_a \rangle$ to node $\langle t+1, l_b \rangle$ for every $1 \leq t < n$ and $l_a, l_b \in \mathcal{L}$. The edge $(\langle t, l_a \rangle, \langle t+1, l_b \rangle)$ is given weight $\phi_t(l_b, l_a|\mathbf{x})$. Finally, we add a start node start and a goal node goal, and for every $l \in \mathcal{L}$, we add edges $(\mathsf{start}, \langle 1, l \rangle)$ with weight $\mathsf{cost}(l|\mathbf{x})$, and edges $(\langle n, l \rangle, \mathsf{goal})$ with weight 0.

Observe that the label sequence $\mathbf{l} = \langle l_1, l_2, \ldots, l_n \rangle$ corresponds to the path $\mathsf{start}, \langle 1, l_1 \rangle, \langle 2, l_2 \rangle, \ldots, \langle n, l_n \rangle, \mathsf{goal}$, and that the weight of this path (sum of edges on this path) is exactly $\mathsf{cost}(\mathbf{l}|\mathbf{x})$. Therefore, the least cost path (of length n) from start to goal corresponds to the desired optimal label sequence. Because there is one-to-one correspondence between label sequences in $\mathcal{Y}$ and path in G from start to goal, we will use the two interchangeably. The label sequence $\mathbf{l}$ corresponds to the path $\mathsf{start}, \langle 1, \mathbf{l}_1 \rangle, \langle 2, \mathbf{l}_2 \rangle, \ldots, \langle n, \mathbf{l}_n \rangle, \mathsf{goal}$. We use $\langle k, l \rangle \in \mathbf{l}$ to denote the fact that the node $\langle k, l \rangle$ is on the path $\mathbf{l} \in \mathcal{Y}$ (in other words, the $\mathbf{l}_k = l$).

We let $\alpha(\langle k, l \rangle|\mathbf{x})$ be the cost of the least weight path from start to $\langle k, l \rangle$. The cost of completion of a node $\langle k, l \rangle$ is the cost of the least weight path from $\langle k, l \rangle$ to

goal, and we denote this by $\beta(\langle k,l\rangle|\mathbf{x})$. Observe that $\alpha(\langle k,l\rangle|\mathbf{x}) + \beta(\langle k,l\rangle|\mathbf{x})$ is the cost of the least weight path from start to goal going through $\langle k,l\rangle$. In other words,

$$\alpha(\langle k,l\rangle|\mathbf{x}) + \beta(\langle k,l\rangle|\mathbf{x}) = \min_{\substack{\mathbf{l}\in\mathcal{L}^n \\ \langle k,l\rangle\in\mathbf{l}}} \Big[\mathsf{cost}(\mathbf{l}|\mathbf{x})\Big]$$

Definition 1. *A function* $\mathsf{lower} : V \to \mathbb{R}$ *is an admissible underestimate if for every* $\langle k,l\rangle$ *we have*

1. $\mathsf{lower}(\langle k,l\rangle|\mathbf{x}) \leq \beta(\langle k,l\rangle|\mathbf{x})$.
2. $\mathsf{lower}(\langle n,l\rangle|\mathbf{x}) = 0$ *for all* $l \in \mathcal{L}$.

Condition (1) requires the function lower to be an *optimistic* estimate of the cost function, while Condition (2) requires the estimate to be 0 for the last states of the path. A well known result (Russel et al. 1995) states that if lower is an admissible underestimate, and we use

$$\mathsf{prio}(\langle k,l\rangle|\mathbf{x}) = \mathsf{cost}(\langle k,l\rangle|\mathbf{x}) + \mathsf{lower}(\langle k,l\rangle|\mathbf{x})$$

as a priority function for priority driven search, then we are guaranteed to find the optimal solution. In principle, the admissible underestimates allow for pruning of parts of the search space, possibly enabling the algorithm to find the optimal solution faster than the dynamic programming approach. However, maintaining the priority queue adds complexity to the algorithm, and hence for the procedure to outperform the dynamic programming approach in practice, the estimate has to be reasonable sharp. For many cases, even when we can find admissible underestimates, the estimates tend to be loose, and hence do not result in a substantial speedup.

We now consider some relaxed notions of admissible underestimates, and examine the effect of their use on priority driven search.

Definition 2. *A function* $\widetilde{\mathsf{lower}} : V \to \mathbb{R}$ *is probably an* (ϵ,δ)*-approximate underestimate if*

$$\widetilde{\mathsf{lower}}(\langle k,l\rangle|\mathbf{x}) \leq \beta(\langle k,l\rangle|\mathbf{x}) + \delta$$

for every $\langle k,l\rangle$ *and for a randomly drawn* $(\mathbf{x},\mathbf{y}) \in \mathcal{X}\times\mathcal{Y}$ *with probability at least* $1-\epsilon$*, and* $\widetilde{\mathsf{lower}}(\langle n,l\rangle|\mathbf{x}) = 0$ *for all* $l \in \mathcal{L}$.

This definition essentially requires $\widetilde{\mathsf{lower}}$ to be very close to being an underestimate most of the time. The hope would be that if we use $\widetilde{\mathsf{lower}}$ to guide our priority driven search, then we would get a solution which is almost optimal most of the time. The following result shows that this is indeed the case.

Lemma 3. *If* $\widetilde{\mathsf{lower}}$ *is probably an* (ϵ,δ)*-approximate underestimate, and we use*

$$\mathsf{prio}(\langle k,l\rangle|\mathbf{x}) = \mathsf{cost}(\langle k,l\rangle|\mathbf{x}) + \widetilde{\mathsf{lower}}(\langle k,l\rangle|\mathbf{x})$$

to guide a priority driven search, then with probability at least $1-\epsilon$*, we find a solution within* δ *of optimal.*

Proof. We will show that if $\widetilde{\mathsf{lower}}$ satisfies the condition for $\mathbf{x}$, then we find a solution within δ of the optimal. From this, the result will follow. So assume that $\widetilde{\mathsf{lower}}$ satisfies the condition for $\mathbf{x}$ and that $c = \mathsf{cost}(\mathbf{l}^{\mathsf{min}}|\mathbf{x})$ is the cost of an optimal solution $\mathbf{l}^{\mathsf{min}}$.

Suppose that $\mathbf{l}$ is the first complete solution that comes off the priority queue, and let us assume that $\mathsf{cost}(\mathbf{l}|\mathbf{x}) > c+\delta$. Since $\widetilde{\mathsf{lower}}$ is an approximate underestimate, we must have $\widetilde{\mathsf{lower}}(\mathbf{l}|\mathbf{x}) = 0$. Therefore, $\mathsf{prio}(\mathbf{l}|\mathbf{x}) = \mathsf{cost}(\mathbf{l}|\mathbf{x}) + \widetilde{\mathsf{lower}}(\mathbf{l}|\mathbf{x}) > c+\delta$. Since $\mathbf{l}^{\mathsf{min}}$ has not yet been pulled off the priority queue, either $\mathbf{l}^{\mathsf{min}}$ or some node $\langle k,l\rangle \in \mathbf{l}^{\mathsf{min}}$ must still be on the priority queue. Since there is a path from $\langle k,l\rangle$ to $\mathbf{l}^{\mathsf{min}}$, we must have $\mathsf{prio}(\langle k,l\rangle|\mathbf{x}) \leq \mathsf{cost}(\mathbf{l}^{\mathsf{min}}|\mathbf{x}) + \delta = c+\delta$. Hence $\mathsf{prio}(\langle k,l\rangle|\mathbf{x}) \leq c+\delta < \mathsf{prio}(\mathbf{l}|\mathbf{x})$, which means that $\langle k,l\rangle$ should have been pulled off the priority queue before $\mathbf{l}$, a contradiction. □

A probabilistic approximate underestimate makes more sense in a statistical learning context than in a classical AI context. When the state graph is generated from a model with no uncertainty, then the optimal solution is clearly the most desirable solution. In contrast for machine learning applications the *models are statistical in nature*, and even the optimal solutions are "incorrect" some percentage of the time. Therefore, if the decoding algorithm fails to come up with the optimal answer for a fraction ϵ of the cases, then the error rate goes up by at most ϵ; the fundamental statistical nature of the algorithm does not change. Hence it makes sense to consider relaxing the requirements on correctness on the decoding algorithm for machine learning algorithms.

Note also that when the model parameters are estimated from small data sets, the difference between two solutions whose costs are very close may not be statistically significant. In these cases, it may make sense to settle for an approximately optimal algorithm, especially if it will result in a large saving in computation. Therefore, by going in for probable approximate underestimates, we allow ourselves to choose priority functions from a much richer set, and hence there is the potential for much faster inference.

Another important consequence of allowing probably approximate underestimates is that we can now consider learning these underestimates. Guaranteed un-

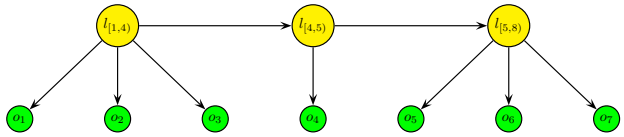

Figure 1: A graphical model for a semi-Markov model. The green circles represent observations, while the yellow circles represent the (hidden) labels/states.

derestimates are often loose, and the effectiveness of a model using these estimates must be verified through experiments. A tighter bound may be obtained by choosing a function that is the "best" underestimate on a finite training set. However, such a function is not guaranteed to be an underestimate on a different data set. We can, however, use generalization bounds from statistical learning theory to show that as long as the class of underestimate functions is not too large, and the training data set is not too small, we will get an approximate underestimate with high probability. Learned underestimates can be much tighter (albeit with some probability of error).

Finally, learned probable approximate underestimates can be applied effectively to a much wider set of statistical models. A common technique for generative models is to estimate completion costs based on a summary of the context (Corazza et al. 1994) or grammar summarization (Klein et al. 2003). In a discriminative setting, where the costs take on a functional form, an estimate based on any meaningful summary of the context will be very loose. This is especially true in cases where the features are deterministically related. In the next section we will show that a learned probably approximate underestimate can be directly applied to a discriminative model.

3 Speeding up semi-Markov Models

As mentioned previously, using a priority driven search algorithm can speed up computation because it can prune away large parts of the search space. However, each individual step is now more expensive because priorities have to be computed, and because the priority order has to be maintained. A very good priority function that is very expensive to compute can well result in an overall decrease in speed. Therefore, it is important to consider both how quickly $\widetilde{\mathsf{lower}}$ can be computed and how sharp it is (i.e., how well it estimates the actual cost).

If $\widetilde{\mathsf{lower}}$ has a similar structure to the cost function, then it is likely that $\widetilde{\mathsf{lower}}$ will be a sharp estimate. On the other hand, the more similar $\widetilde{\mathsf{lower}}$ and cost are, the more similar their computational complexity is, negating the benefit of the the priority driven search. In this section, we show how the cost function of a linear-chain semi-Markov model (defined below) can be approximated using the cost function of a linear-chain Markov model.

Linear-chain Conditional Markov Models (CMMs) and semi-Markov CMMs are discriminative models used for labeling sequence data. The input to these models are sequences of observations $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$, and the model produces a sequence of labels drawn from the label sequence $\mathcal{L}$. Both models assign costs (probabilities) to label sequences conditional on the input (observation) sequence. A widely used example of a CMM is a Conditional Random Field. However, while a CRF assigns a probability to a label sequence $\langle l_1, l_2, \ldots, l_n \rangle$ (conditional on the input observation sequence $\mathbf{x}$), a CMM can be more general, and assign a score which can be based on a general loss function, the margin to a separating surface or on the number of votes from an ensemble. The cost that a CMM assigns to the label sequence $\langle l_1, l_2, \ldots, l_n \rangle$ (conditional on the input observation $\mathbf{x}$) is given by

$$\mathsf{cost}^{\mathsf{cmm}}(\mathbf{l}|\mathbf{x}) = \sum_{t=2}^{n} \sum_{f \in \mathcal{F}_{\mathsf{cmm}}} \lambda_f f(t, l_{t-1}, l_t, \mathbf{x})$$

Note that this cost could have been derived from the log-probability assigned by a linear-chain CRF whose underlying linear graphical model has (maximal) cliques with potential functions of the form

$$\phi_t\left(l_{t-1}, l_t, \mathbf{x}\right) = \sum_{f \in \mathcal{F}_{\mathsf{cmm}}} -\lambda_f f\left(t, l_{t-1}, l_t, \mathbf{x}\right)$$

A semi-Markov model (Sarawagi et al. 2004) is one where each hidden state is associated with a set of observations. These models provide a richer mechanism for modeling dependencies between observations associated with a single state. While the most natural language for describing such models is generative, the framework below applied to discriminative models as well.

In a semi-Markov model, a sequence of m-labels (for $m \leq n$) are generated, and for each label, one or more observations are generated. We denote by the segment/label sequence $\langle l_{[s_1:s_2)}, l_{[s_2,s_3)}, \ldots, l_{[s_{m-1},s_{m+1})} \rangle$ the fact that label $l_{[s_i,s_{i+1})}$ generates the observations $o_{s_i}, o_{s_i+1}, \ldots, o_{s_{i+1}-1}$. Here $1 = s_1 < s_2 < \cdots < s_{m+1} = n$ denote the segment boundaries.

The cost that the semi-Markov CMM assigns to the segment/label sequence

$\langle l_{[s_1:s_2)}, l_{[s_2,s_3)}, \ldots, l_{[s_{m-1},s_{m+1})}\rangle$ is given by

$$\begin{aligned}&\mathsf{cost}^{\mathsf{scmm}}\left(\langle l_{[s_1:s_2)}, l_{[s_2,s_3)}, \ldots, l_{[s_m,s_{m+1})}\rangle|\mathbf{x}\right)\\ &= \sum_{t=1}^{m-1} \psi_{(s_{t+1},s_{t+2})}(l_{[s_t,s_{t+1})}, l_{[s_{t+1},s_{t+2})}, \mathbf{x})\end{aligned}$$

where the potentials ψ_t are given by

$$\begin{aligned}&\psi_{(s_t,s_{t+1})}(l_{[s_{t-1},s_t)}, l_{[s_t,s_{t+1})}, \mathbf{x})\\ &= \sum_{f\in\mathcal{F}_{\mathsf{scmm}}} \lambda_f f\left(s_t, s_{t+1}, l_{[s_{t-1},s_t)}, l_{[s_t s_{t+1})}, \mathbf{x}\right)\end{aligned}$$

Each semi-Markov CMM feature $f \in \mathcal{F}_{\mathsf{scmm}}$ can be a function of the observations, the current segment $[s_{t+1}, s_{t+2})$, and the current and previous labels $l_{[s_{t-1},s_t)}$ and $l_{[s_t s_{t+1})}$. Note that $\mathcal{F}_{\mathsf{scmm}}$, is richer than $\mathcal{F}_{\mathsf{cmm}}$, the set of features available to the CMM, because the semi-Markov features $f\left(s_t, s_{t+1}, l_{[s_t,s_{t+1})}, l_{[s_{t+1}s_{t+2})}, \mathbf{x}\right)$ can also depend on the entire segments $[s_t, s_{t+1})$. As a result, semi-Markov CMMs typically yield higher accuracies than CMMs. However, the decoding time for these models is $O(n^2)$. In (Sarawagi 2006), a technique for improving the efficiency of inference in semi-Markov CMMs is presented by reorganizing the clique potential computation is presented. This technique is completely orthogonal to our technique, and both can be applied to yield greater speedups. In this section, we discuss how we can use $\mathsf{cost}_{\mathsf{cmm}}$ as a probable approximate underestimate for $\mathsf{cost}_{\mathsf{scmm}}$.

We start by describing the graph $G_S = (V_S, E_S)$ corresponding to the search problem for semi-Markov Models. The node set $V_S = \{\langle s, r, l\rangle : 1 \le s < r \le n, l \in \mathcal{L}\} \cup \{\mathsf{start}, \mathsf{goal}\}$. So, now each node corresponds to the time range $[s, r)$ and label l. There are edges between nodes $\langle s, r, l_1\rangle$ and $\langle r, q, l_2\rangle$ for $1 \le s < r < q \le n$, and this edge has cost $\sum_{f\in\mathcal{F}_{\mathsf{scmm}}} f(s, r, t, l_1, l_2, \mathbf{x})$. A common pruning step is to place a limit W on the length of the largest segment, only allowing nodes $\langle s, r, l\rangle$ which satisfy $r - s < W$. In this case, the decoding time required for the dynamic programming solution reduces to $O(n \cdot W)$. However, it is often the case that $W \in O(n)$, and hence this may not result in substantial savings.

The cost of completion (cost to goal) is the cost of the least cost path to the goal. Any function which is less than this can serve to be an optimistic estimate to the goal. Let us denote by $\beta_{\mathsf{scmm}}(\langle s_k, s_{k+1}, l_k\rangle|\mathbf{x})$ the cost of the least cost path from $\langle s_k, s_{k+1}, l_k\rangle$ to goal. A completion path is of the form $\langle s_k, s_{k+1}, l_k\rangle, \langle s_{k+1}, s_{k+2}, l_{k+1}\rangle, \ldots, \langle s_m, s_{m+1}, l_m\rangle$ where $s_{m+1} = n$, and its cost is given by

$$\sum_{t=k}^{m-1} \psi_{(s_t,s_{t+1},s_{t+2})}(l_{[s_t,s_{t+1})}, l_{[s_{t+1},s_{t+2})}|\mathbf{x})$$

So $\beta_{\mathsf{scmm}}(\langle s_k, s_{k+1}, l_k\rangle)$ is the least value of all costs of the above form (this can be computed by dynamic programming in polynomial time). Now, we want to estimate a function that will serve as an probabilistic approximate underestimate for β. Given the similarity in the forms of the cost functions of CMMs, $\mathsf{cost}_{\mathsf{cmm}}$, and the cost functions of the semi-Markov Model, $\mathsf{cost}_{\mathsf{scmm}}$, it makes sense to see if we can use $\mathsf{cost}_{\mathsf{cmm}}$ to generate the desired probabilistic approximate underestimate.

Given a CMM search graph with nodes $\langle t, l\rangle$ and a semi-Markov Model with nodes $\langle s, r, l\rangle$, we can map $\langle s, r, l\rangle \mapsto \langle s, l\rangle$ (this is a many-to-one mapping). It is then natural to try to estimate $\beta_{\mathsf{scmm}}(\langle s, r, l\rangle)$ using $\beta_{\mathsf{cmm}}(\langle s, l\rangle)$. For this, we want

$$\beta_{\mathsf{cmm}}(\langle s, l\rangle|\mathbf{x}) \le \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}) + \delta$$

for every node $\langle s, r, l\rangle \in V_S$ for all but an ϵ fraction of the input/output pairs. Therefore, we seek (the parameters of) a CMM, which satisfies this condition. Observe that is a circularity in the requirements here. The optimal path used for completion in the CMM depends on the $\mathsf{cost}_{\mathsf{cmm}}$. But we want to pick $\mathsf{cost}_{\mathsf{cmm}}$ based on the optimal completion path. We use the following trick to resolve this circularity. Let $\boldsymbol{l} = \langle z_1, z_2, \ldots, z_n\rangle$ be the label sequence generated by a computationally cheap classifier (for our experiments, we used a classifier obtained by boosting small-depth decision trees). For any node, $\langle s, l\rangle \in V$, we can generate a completion path $P(\mathbf{z}, \langle s, l\rangle) = \langle s, l\rangle, \langle s+1, z_{s+1}\rangle, \ldots, \langle s_n, z_n\rangle$. We pick $\mathsf{cost}_{\mathsf{cmm}}$ to satisfy

$$\mathsf{cost}_{\mathsf{cmm}}(P(\mathbf{z}, \langle s, l\rangle)) \le \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}) + \delta \qquad (1)$$

Since the cost of the optimal path is less than the cost of any fixed path, we must have

$$\begin{aligned}\beta_{\mathsf{cmm}}(\langle s, l\rangle|\mathbf{x}) &\le \mathsf{cost}_{\mathsf{cmm}}(P(\mathbf{z}, \langle s, l\rangle)\\ &\le \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}) + \delta\end{aligned}$$

Therefore, if we can find $\mathsf{cost}_{\mathsf{cmm}}$ probably satisfying this condition, we can use it as an probabilistic approximate underestimate. This condition translates to

$$\begin{aligned}\sum_{t=s}^{n} \sum_{f\in\mathcal{F}_{\mathsf{cmm}}} \lambda_f f(t, \ell_{t-1}, \ell_t|\mathbf{x}) &= \sum_{t=s}^{n} \phi_t(t, \ell_{t-1}, \ell_t|\mathbf{x})\\ &= \mathsf{cost}_{cmm}(P(\mathbf{z}, \langle s, l\rangle)|\mathbf{x})\\ &\le \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}) + \delta\end{aligned}$$

where

$$\ell_t = \begin{cases} l & \text{if } s \le t < r\\ z_t & \text{otherwise}\end{cases}$$

The parameters (variables) that we can pick are $\{\lambda_f\}_{f\in\mathcal{F}_{\mathsf{cmm}}}$. In the next section, we show how these values can be estimated from data as the solution to an optimization problem.

4 Optimizing CMM estimates

Suppose that we are given a collection of *unlabeled* sequences $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, and a trained semi-Markov model. We wish to estimate the parameters of a CMM $\{\lambda_f\}_{f\in\mathcal{F}_{\mathsf{cmm}}}$ so that the resulting cost function satisfies Equation (1) given in the previous section. For each sequence $\mathbf{x}^{(i)}$, let $\mathbf{z}^{(i)}$ be the output label sequence from a computationally cheap classifier. For each example $1 \le i \le N$, and for each state $\langle s, r, l\rangle$, we let

$$\delta_{(i,s,r,l)} = \sum_{t=s}^{n} \sum_{f\in\mathcal{F}_{\mathsf{cmm}}} \lambda_f f(t, \ell_{t-1}, \ell_t|\mathbf{x}^{(i)}) - \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}^{(i)})$$

By taking

$$\delta_{(i,s,r,l)} \le \delta - \mu$$

(where $\mu \ge 0$ is analogous to a margin) we satisfy Equation (1). We omit the details due lack of space, but generalization bounds, much like those obtained for SVMs can be obtained for the underestimate as well. The reason for introducing the "margin" μ is to enable us to prove generalization bounds (so that the resulting solution which is an approximate underestimate on the test set is also an approximate underestimate on the training set). Larger value of μ and N makes it more likely that the generated CMM will also be an underestimate on the test set. However, smaller values of μ are desirable because this allows for tighter bounds.

The value of $\beta_{scrf}(\langle s, r, l\rangle|\mathbf{x}^{(i)})$ can be computed for all values of $\langle s, r, l\rangle$ by simply running the dynamic programming algorithm and then reading the values off the table used to store the partial results. While the premise of this paper has been that this is an expensive operation, it only has to be done offline, and only once (per example). Similarly, the values of $f(t, \ell_{t-1}, \ell_t|\mathbf{x}^{(i)})$ can be computed for for all the examples once offline, and hence the system of inequalities can be set up.

Observe that $\left|\delta_{(i,s,r,l)}\right|$ measures the inexactness of the underestimate. The smaller this quantity, the better the estimate. If this quantity is negative, then it is an underestimate, and if it is positive, it is an overestimate. The constraint ensures it never overestimates by more than $\overline{\delta}$. Since we seek to make the underestimate as sharp as possible, we want to minimize $\left|\delta_{(i,s,r,l)}\right|$. Therefore, the objective function that is used for the optimization is

$$\lambda \cdot \|\mathbf{f}\| + \sum \left|\delta_{(i,s,r,l)}\right|$$

The term $\lambda \cdot \|\mathbf{f}\|$ acts as a regularizer. Both the ℓ_1 and the ℓ_2 norms can be used (both yield (different) generalization bounds because for a finite dimensional space, all norms differ by at most a constant). The advantage of using the ℓ_1 norm is that it often yields more sparse solutions, yielding added speedups by discarding features whose coefficients are zero. When using the ℓ_1 norm, the resulting problem is a linear programming problem. When using the ℓ_2 norm, the resulting problem is a quadratic programming problem (similar to the standard SVM problem).

Therefore, in this formulation, there are at most $|\mathcal{F}_{\mathsf{cmm}}| + n^2 \cdot N\,|\mathcal{L}|$ variables, and at most $n^2 \cdot N\,|\mathcal{L}|$ inequalities (plus the box constraints). Since the procedure only requires unlabeled examples, we can potentially feed it a tremendous amount of data. Since the size of the optimization problem (both the number of variables and the number of constraints) grows linearly with the number of examples N, the problem as formulated above very rapidly exhausts the capacity of most optimization procedures. We now discuss two tricks which can be used to extend the range of these procedures.

4.1 Generating sparse problem formulations

Representing n_1 equations/inequalities in n_2 variables requires $O(n_1 \cdot n_2)$ storage when using a dense matrix representation. When the problem can be formulated so that the equations/inequalities are sparse (so each inequality involves only a small number of variables), and if the optimization solver is able to exploit the sparsity of the formulation, we can get both efficient representations and efficient solution procedures. This allows us to store larger problem in memory, and solve problems more quickly. A slight modification of the formulation we presented allows us to reduce the number of non-zero entries significantly. For a fixed example $\mathbf{x}^{(i)}$, consider the set of equations:

$$\delta_{(i,s,r,l)} = \sum_{t=s}^{n} \sum_{f\in\mathcal{F}_{\mathsf{cmm}}} \lambda_f f(t, \ell_{t-1}, \ell_t|\mathbf{x}^{(i)}) - \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}^{(i)})$$

Let

$$\gamma_{(i,t)} = \sum_{f\in\mathcal{F}_{\mathsf{cmm}}} \lambda_f f(t, z_{t-1}, z_t|\mathbf{x}^{(i)})$$

Then we may write

$$\delta_{(i,s,r,l)} = \sum_{t=s}^{n} \gamma_{(i,t)} - \beta_{\mathsf{scmm}}(\langle s, r, l\rangle|\mathbf{x}^{(i)})$$

Observe that these two systems of equations are equivalent, except the second formulation has substantially fewer non-zero entries even though we have added a few extra variables ($n \cdot N$ extra variables).

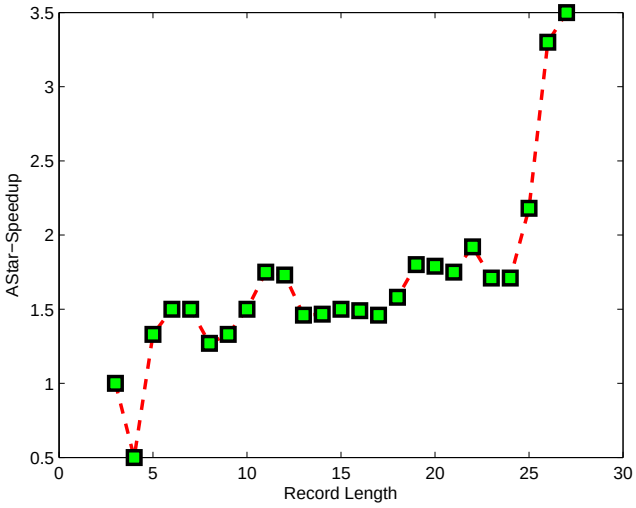

Figure 2: Variation of the A*-decoding speedup over Viterbi with the number of tokens in the sequence

4.2 Discarding inessential inequalities

Another way of reducing the memory footprint, and speeding up the solution is to discard several of the inequalities completely. To see why we might be able to get away with doing this, let $\mathbf{y}^{(i)}$ be the optimal label sequence for the input sequence $\mathbf{z}^{(i)}$. Then as long as Equation (1) holds for all the nodes on the optimal label sequence, then the result of Lemma 3 still holds. In fact, we would prefer that nodes that are not part of the optimal sequence get very pessimistic estimates, as this ensures that they are not explored further, increasing the speed of the search algorithm. Therefore, if we discard the inequalities corresponding to the nodes which are not part of the optimal label sequence, then while the CMM so generated will no longer be an approximate probabilistic underestimate, it is still guaranteed to produce approximately optimal solutions on the training data. In practice this seems to work quite well, producing very little deterioration on the test sets.

5 Experiments

For our experiments, we chose the problem of automatically labeling contact fragments. The data was collected from web pages and email, and was hand-labeled with 24 labels (FirstName, MiddleName, LastName, NickName, Suffix, Title, JobTitle, CompanyName, Department, AddressLine, City, State, Country, PostalCode, HomePhone, Fax, CompanyPhone, DirectCompanyPhone, MobilePhone, Pager, Email, InstantMessage, WebPage). We used 3100 contact records to train the semi-Markov CMM and the CMM underestimate. We used a different set of 1487 contact records to evaluate the quality (accuracy and speed) of the various decoding algorithms. The features for the CMM included various simple regular expressions expressing concepts like IsCapitalized, AllCaps, IsDigit, Numeric, ContainsDash, EndsInPeriod, ConstainsAtSign, etc. In addition there are 9 lexicon lists including: LastNames, FirstNames, States, Cities, Countries, JobTitles, CompanyNameComponents, Titles, StreetNameComponents. The semi-Markov model used several additional features including segment TFIDF features, segment length features, and segment boundary features (such as AllOnSameLine, EndsOnNewLine etc.). Our baseline is the the voted perceptron conditional Markov model described in (Collins 2002). The form of the Collins model is very similar to the more well known CRF (the decoding algorithms are identical). While experiments show that the two systems are quite competitive, the implementation of the learning algorithm for the Collins model is much simpler. Our results using the Collins model are very similar to earlier results obtained by Kristjansson et. al on a similar problem (Kristjansson et. al 2004).

The computationally cheap classifier that is used for generating the path along which the CMM estimate is computed is obtained by boosting decision trees. This classifier uses the same regular expression and lexicon features that the CMM uses. The final classifier had a total of 46 decision trees of depth 3, and the token accuracy of this classifier is 0.861.

Table 1 compares the various decoding algorithms on our test set. As can be seen, the priority based search procedure achieves practically the same accuracy as the Viterbi decoding algorithm, but takes about one third the time to parse our data set. Figure 2 shows how the speedup varies with the length of the sequence. It can be seen that when the sequence length is small, the priority based search procedure offers no significant advantage over the Viterbi decoding algorithm. In fact, for very small sequence length, the priority based search procedure can actually take more time (due to the overhead of maintaining the priority queue). However, for larger values of sequence length, the priority based search algorithm can reduce the computation time by a factor of 3.5. For the data sets that we used to train the underestimate, the LP that is used to compute the underestimate takes a few hours to solve when we use the speedup techniques described in Subsections 4.1 and 4.2.

	CMM	Viterbi-SCMM	A*-SCMM
TER	0.896	0.927	0.923
RER	0.487	0.642	0.638
Decode Time	1	7.17	2.26

Table 1: Relative performance of CMM and Semi-Markov Model with Viterbi and A* Decoding with respect to accuracy (TER = Token Error Rate, RER = Record Error Rate) and speed. The decoding speeds have been normalized so that CMM time = 1.

6 Conclusions and Future Work

We introduced the notion of a probable approximate underestimate, which we show can be used in a priority-driven search algorithm to produce an A*-like algorithm which is guaranteed to produce a near-optimal solutions with high probability. We show that this probable approximate underestimate can be learned from from unlabeled data, and show that this results in substantial speedups for semi-Markov Models. This technique results in greater speedups on inputs for which the Viterbi algorithm takes more time, and hence we expect that this will be very effective for decoding more complicated models. We are currently experimenting with alternative optimization algorithms and extensions to more general discriminative grammars.

References

A. Corazza, R. De Mori, R. Gretter and G. Satta (1994). Optimal Probabilistic Evaluation Functions for Search Controlled by Stochastic Context-Free Grammars. IEEE Transactions on Pattern Analysis and Machine Intelligence (16-4).

S. J. Russel and P. Norvig (1995). Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, NJ.

S. A. Caraballo and E. Charniak (1997). Figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24, pp. 275–298.

J. Goodman (1997). Global thresholding and multiple-pass parsing. In *Proceedings of Empirical Methods in Natural Language Processing*, pp. 11-25.

E. Charniak, S. Goldwater and M. Johnson (1998). Edge-based best-first chart parsing. In *Proceedings of the sixth workshop on very large corpora*, pp. 127–133.

R. Durbin, S. R. Eddy, A. Krogh and G. Mitchison (1999). Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press.

V. Goel and W. J. Bryne (1999). Taks dependent loss functions in speech recognition: A-start search over recognition lattices. In Eurospeech, pp. 1243–1246.

A. Ratnaparkhi (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 34. pp. 151–175

B. Roark (2001). Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27, pp. 249–276.

M. Collins (2002). Discriminative training methods for hidden Markov models : Theory and experiments with perceptron algorithms In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP02).*

D. Klein and C. Manning (2003). A* parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of HLT-NAACL.*

Z. Weinberg and L. Ruzzo (2006). Sequence-based heuristics for faster annotation of non-coding RNA families. *Bioinformatics*, 22(1), pp. 35–39.

M. Collins and B. Roark (2004). Incremental parsing with the perceptron algorithm, In *Proceedings of the annual meeting of the Association for Computational Linguistics.*

T. Kristjansson, A. Culotta, P. Viola and A. McCallum (2004). Interactive Information Extraction with Constrained Conditional Random Fields. In *Proceedings of the 19th International Conference on Artificial intelligence (AAAI).*

S. Sarawagi and W. W. Cohen (2004). Semi-Markov conditional random fields for information extraction. In *Proceedings of Neural Information Processing Systems.*

H. Daumé III and D. Marcu (2005). Learning as search optimization: approximate large margin methods for structured prediction, In *Proceedings of the 22nd International conference on Machine Learning.*

P. Viola and M. Narasimhan (2005). Discriminative grammars for information extraction, In *Proceedings of Annual ACM SIGIR Conference.*

N. Ratliff, D. Bradley, J. A. Bagnell and J. Chestnutt (2007). Boosting structured prediction for imitation learning, In *Proceeding sof Neural Information Processing Systems.*

S. Sarawagi (2006). Efficient inference on sequence segmentation models. In *Proceedings of International Conference on Machine Learning.*