Iterative Viterbi A* Algorithm for K-Best Sequential Decoding

Zhiheng Huang[†], Yi Chang, Bo Long, Jean-Francois Crespo[†],
Anlei Dong, Sathiya Keerthi and Su-Lin Wu
Yahoo! Labs
701 First Avenue, Sunnyvale
CA 94089, USA
{zhiheng_huang, jfcrespo}@yahoo.com[†]
{yichang,bolong,anlei,selvarak,sulin}@yahoo-inc.com

Abstract

Sequential modeling has been widely used in a variety of important applications including named entity recognition and shallow parsing. However, as more and more real time large-scale tagging applications arise, decoding speed has become a bottleneck for existing sequential tagging algorithms. In this paper we propose 1-best A*, 1-best iterative A*, k-best A* and k-best iterative Viterbi A* algorithms for sequential decoding. We show the efficiency of these proposed algorithms for five NLP tagging tasks. In particular, we show that iterative Viterbi A* decoding can be several times or orders of magnitude faster than the state-of-the-art algorithm for tagging tasks with a large number of labels. This algorithm makes real-time large-scale tagging applications with thousands of labels feasible.

1 Introduction

Sequence tagging algorithms including HMMs (Rabiner, 1989), CRFs (Lafferty et al., 2001), and Collins's perceptron (Collins, 2002) have been widely employed in NLP applications. Sequential decoding, which finds the best tag sequences for given inputs, is an important part of the sequential tagging framework. Traditionally, the Viterbi algorithm (Viterbi, 1967) is used. This algorithm is quite efficient when the label size of problem modeled is low. Unfortunately, due to its $O(TL^2)$ time complexity, where T is the input token size and L is the label size, the Viterbi decoding can become prohibitively slow when the label size is large (say, larger than 200).

It is not uncommon that the problem modeled consists of more than 200 labels. The Viterbi algorithm cannot find the best sequences in tolerable response time. To resolve this, Esposito and Radicioni (2009) have proposed a Carpediem algorithm which opens only necessary nodes in searching the best sequence. More recently, Kaji et al. (2010) proposed a staggered decoding algorithm, which proves to be very efficient on datasets with a large number of labels.

What the aforementioned literature does not cover is the k-best sequential decoding problem, which is indeed frequently required in practice. For example to pursue a high recall ratio, a named entity recognition system may have to adopt k-best sequences in case the true entities are not recognized at the best one. The k-best parses have been extensively studied in syntactic parsing context (Huang, 2005; Pauls and Klein, 2009), but it is not well accommodated in sequential decoding context. To our best knowledge, the state-of-the-art k-best sequential decoding algorithm is *Viterbi A** [1]. In this paper, we generalize the *iterative process* from the work of (Kaji et al., 2010) and propose a k-best sequential decoding algorithm, namely *iterative Viterbi A**. We show that the proposed algorithm is several times or orders of magnitude faster than the state-of-the-art in all tagging tasks which consist of more than 200 labels.

Our contributions can be summarized as follows. (1) We apply the A* search framework to sequential decoding problem. We show that A* with a proper heuristic can outperform the classic Viterbi decoding. (2) We propose 1-best A*, 1-best iterative A* decoding algorithms which are the second and third fastest decoding algorithms among the five decoding algorithms for comparison, although there is a significant gap to the fastest 1-best decoding algorithm. (3) We propose k-best A* and k-best iterative Viterbi A* algorithms. The latter is several times or orders of magnitude faster than the state-of-the-art

[1] Implemented in both CRFPP (http://crfpp.sourceforge.net/) and LingPipe (http://alias-i.com/lingpipe/) packages.

k-best decoding algorithm. This algorithm makes real-time large-scale tagging applications with thousands of labels feasible.

2 Problem formulation

In this section, we formulate the sequential decoding problem in the context of perceptron algorithm (Collins, 2002) and CRFs (Lafferty et al., 2001). All the discussions apply to HMMs as well. Formally, a perceptron model is

$$f(\mathbf{y}, \mathbf{x}) = \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t), \quad (1)$$

and a CRFs model is

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\{\sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\}, \quad (2)$$

where $\mathbf{x}$ and $\mathbf{y}$ is an observation sequence and a label sequence respectively, t is the sequence position, T is the sequence size, f_k are feature functions and K is the number of feature functions. θ_k are the parameters that need to be estimated. They represent the importance of feature functions f_k in prediction. For CRFs, $Z(\mathbf{x})$ is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\{\sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\}. \quad (3)$$

If $\mathbf{x}$ is given, the decoding is to find the best $\mathbf{y}$ which maximizes the score of $f(\mathbf{y}, \mathbf{x})$ for perceptron or the probability of $p(\mathbf{y}|\mathbf{x})$ for CRFs. As $Z(\mathbf{x})$ is a constant for any given input sequence $\mathbf{x}$, the decoding for perceptron or CRFs is identical, that is,

$$\arg\max_{\mathbf{y}} f(\mathbf{y}, \mathbf{x}). \quad (4)$$

To simplify the discussion, we divide the features into two groups: unigram label features and bigram label features. Unigram features are of form $f_k(y_t, \mathbf{x}_t)$ which are concerned with the current label and arbitrary feature patterns from input sequence. Bigram features are of form $f_k(y_t, y_{t-1}, \mathbf{x}_t)$ which are concerned with both the previous and the current labels. We thus rewrite the decoding problem as

$$\arg\max_{\mathbf{y}} \sum_{t=1}^{T} (\sum_{k=1}^{K_1} \theta_k^1 f_k^1(y_t, \mathbf{x}_t) + \sum_{k=1}^{K_2} \theta_k^2 f_k^2(y_t, y_{t-1}, \mathbf{x}_t)). \quad (5)$$

For a better understanding, one can interpret the term $\sum_{k=1}^{K_1} \theta_k^1 f_k^1(y_t, \mathbf{x}_t)$ as node y_t's score at position t, and interpret the term $\sum_{k=1}^{K_2} \theta_k^2 f_k^2(y_t, y_{t-1}, \mathbf{x}_t)$ as edge (y_{t-1}, y_t)'s score. So the sequential decoding problem is cast as a max score pathfinding problem[2]. In the discussion hereafter, we assume scores of nodes and edges are pre-computed (denoted as $n(y_t)$ and $e(y_{t-1}, y_t)$), and we can thus focus on the analysis of different decoding algorithms.

3 Background

We present the existing algorithms for both 1-best and k-best sequential decoding in this section. These algorithms serve as basis for the proposed algorithms in Section 4.

3.1 1-Best Viterbi

The Viterbi algorithm is a classic dynamic programming based decoding algorithm. It has the computational complexity of $O(TL^2)$, where T is the input sequence size and L is the label size[3]. Formally, the Viterbi computes $\alpha(y_t)$, the best score from starting position to label y_t, as follows.

$$\max_{y_{t-1}}(\alpha_{y_{t-1}} + e(y_{t-1}, y_t)) + n(y_t), \quad (6)$$

where $e(y_{t-1}, y_t)$ is the edge score between nodes y_{t-1} and y_t, $n(y_t)$ is the node score for y_t. Note that the terms $\alpha_{y_{t-1}}$ and $e(y_{t-1}, y_t)$ take value 0 for $t = 0$ at initialization. Using the recursion defined above, we can compute the highest score at end position $T - 1$ and its corresponding sequence. The recursive computation of α_{y_t} is denoted as forward pass since the computing traverses the lattice from left to right. Conversely, the backward pass computes β_{y_t} as the follows.

$$\max_{y_{t+1}}(\beta_{y_{t+1}} + e(y_t, y_{t+1}) + n(y_{t+1})). \quad (7)$$

Note that $\beta y_{T-1} = 0$ at initialization. The max score can be computed using $\max_{y_0}(\beta_0 + n(y_0))$. We can use either forward or backward pass to compute the best sequence. Table 1 summarizes the computational complexity of all decoding algorithms including Viterbi, which has the complexity of TL^2 for both best and worst cases. Note that N/A means the decoding algorithms are not applicable (for example, iterative Viterbi is not applicable to k-best decoding). The proposed algorithms (see Section 4) are highlighted in bold.

3.2 1-Best iterative Viterbi

Kaji et al. (Kaji et al., 2010) presented an efficient sequential decoding algorithm named *staggered decoding*. We use the name *iterative Viterbi* to describe

[2] With the constraint that the path consists of one and only one node at each position.

[3] We ignore the feature size terms for simplicity.

this algorithm for the reason that the iterative process plays a central role in this algorithm. Indeed, this iterative process is generalized in this paper to handle k-best sequential decoding (see Section 4.4).

The main idea is to start with a coarse lattice which consists of both *active* labels and *degenerate* labels. A label is referred to as an *active label* if it is not grouped (e.g., all labels in Fig. 1 (a) and label A at each position in Fig. 1 (b)), and otherwise as an *inactive label* (i.e., dotted nodes). The new label, which is made by grouping the inactive labels, is referred to as a *degenerate label* (i.e., large nodes covering the dotted ones). Fig. 1 (a) shows a lattice which consists of active labels only and (b) shows a lattice which consists of both active and degenerate ones. The score of a degenerate label is the max score of inactive labels which are included in the degenerate label. Similarly, the edge score between a degenerate label z and an active label y' is the max edge score between any inactive label $y \in z$ and y', and the score of two degenerate labels z and z' is the max edge score between any inactive label $y \in z$ and $y' \in z'$. Using the above definitions, the best sequence derived from a degenerate lattice would be the upper bound of the sequence derived from the original lattice. If the best sequence does not include any degenerate labels, it is indeed the best sequence for the original lattice.

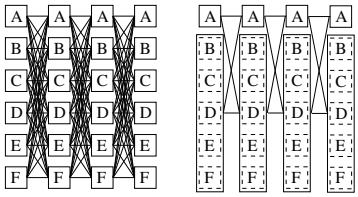

Figure 1: (a) A lattice consisting of active labels only. (b) A lattice consisting of both active labels and degenerate ones. Each position has one active label (A) and one degenerate label (consisting of B, C. D, E, and F).

The pseudo code for this algorithm is shown in Algorithm 1. The lattice is initialized to include one active label and one degenerate label at each position (see Figure 1 (b)). Note that the labels are ranked by the probabilities estimated from the training data. The Viterbi algorithm is applied to the lattice to find the best sequence. If the sequence consists of active labels only, the algorithm terminates and returns such a sequence. Otherwise, the lower bound lb[4] of the active sequence in the lattice is updated and the lattice is expanded. The lower bound can be initialized to the best sequence score using a beam search (with beam size being 1). After either a forward or a backward pass, the lower bound is assigned with the best active sequence score $best(lattice)$[5] if the former is less than the latter. The expansion of lattice ensures that the lattice has twice active labels as before at a given position. Figure 2 shows the column-wise expansion step. The number of active labels in the column is doubled only if the best sequence of the degenerate lattice passes through the degenerate label of that column.

Algorithm 1 Iterative Viterbi Algorithm

```
lb = best score from beam search
init lattice
for i=0;;i++ do
    if i %2 == 0 then
        y = forward()
    else
        y = backward()
    end if
    if y consists of active labels only then
        return y
    end if
    if lb < best(lattice) then
        lb = best(lattice)
    end if
    expand lattice
end for
```

Algorithm 2 Forward

```
for i=0; i < T; i++ do
    Compute α(y_i) and β(y_i) according to Equations (6) and (7)
    if α(y_i) + β(y_i) < lb then
        prune y_i from the current lattice
    end if
end for
Node b = arg max_{y_{T-1}} α(y_{T-1})
return sequence back tracked by b
```

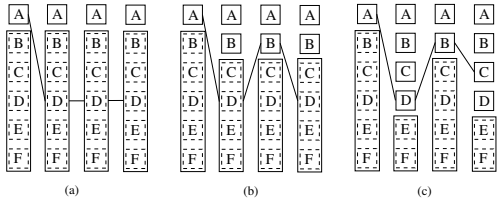

Figure 2: Column-wise lattice expansion: (a) The best sequence of the initial degenerate lattice, which does not pass through the degenerate label in the first column. (b) Column-wise expansion is performed and the best sequence is searched again. Notice that the active label in the first column is not expanded. (c) The final result.

Algorithm 2 shows the forward pass in which the node pruning is performed. That is, for any node, if the best score of sequence which passes such a node is less than the lower bound lb, such a node is removed from the lattice. This removal is safe as such a node does not have a chance to form an optimal sequence. It is worth noting that, if a node is removed, it can no longer be added into the lattice.

[4]The maximum score of the active sequences found so far.

[5]We do not update the lower bound lb if we cannot find an active sequence.

This property ensures the efficiency of the iterative Viterbi algorithm. The backward pass is similar to the forward one and it is thus omitted.

The alternative calls of forward and backward passes (in Algorithm 1) ensure the alternative updating/lowering of node forward and backward scores, which makes the node pruning in either forward pass (see Algorithm 2) or backward pass more efficient. The lower bound lb is updated once in each iteration of the main loop in Algorithm 1. While the forward and backwards scores of nodes gradually decrease and the lower bound lb increases, more and more nodes are pruned.

The iterative Viterbi algorithm has computational complexity of T and TL^2 for best and worst cases respectively. This can be proved as follows (Kaji et al., 2010). At the m-th iteration in Algorithm 1, iterative Viterbi decoding requires order of $T4^m$ time because there are 2^m active labels (plus one degenerate label). Therefore, it has $\sum_{i=0}^{m} T4^i$ time complexity if it terminates at the m-th iteration. In the best case in which $m = 0$, the time complexity is T. In the worst case in which $m = \lceil \log_2 L \rceil - 1$ ($\lceil . \rceil$ is the ceiling function which maps a real number to the smallest following integer), the time complexity is order of TL^2 because $\sum_{i=0}^{\lceil \log_2 L \rceil - 1} T4^i < 4/3TL^2$.

3.3 1-Best Carpediem

Esposito and Radicioni (2009) have proposed a novel 1-best[6] sequential decoding algorithm, *Carpediem*, which attempts to open only necessary nodes in searching the best sequence in a given lattice. Carpediem has the complexity of $TL \log L$ and TL^2 for the best and worst cases respectively. We skip the description of this algorithm due to space limitations. Carpediem is used as a baseline in our experiments for decoding speed comparison.

3.4 K-Best Viterbi

In order to produce k-best sequences, it is not enough to store 1-best label per node, as the k-best sequences may include suboptimal labels. The k-best sequential decoding gives up this 1-best label memorization in the dynamic programming paradigm. It stores up to k-best labels which are necessary to form k-best sequences. The k-best Viterbi algorithm thus has the computational complexity of KTL^2 for both best and worst cases.

Once we store the k-best labels per node in a lattice, the k-best Viterbi algorithm calls either the forward or the backward passes just in the same way as the 1-best Viterbi decoding does. We can compute the k highest score at the end position $T - 1$ and the corresponding k-best sequences.

3.5 K-Best Viterbi A*

To our best knowledge the most efficient k-best sequence algorithm is the Viterbi A* algorithm as shown in Algorithm 3. The algorithm consists of one forward pass and an A* backward pass. The forward pass computes and stores the Viterbi forward scores, which are the best scores from the start to the current nodes. In addition, each node stores a *backlink* which points to its predecessor.

The major part of Algorithm 3 describes the backward A* pass. Before describing the algorithm, we note that each node in the agenda represents a sequence. So the operations on nodes (push or pop) correspond to the operations on sequences. Initially, the L nodes at position $T - 1$ are pushed to an agenda. Each of the L nodes n_i, $i = 0, \ldots, L - 1$, represents a sequence. That is, node n_i represents the best sequence from the start to itself. The best of the L sequences is the globally best sequence. However, the i-th best, $i = 2, \ldots, k$, of the L sequence may not be the globally i-th best sequence. The priority of each node is set as the score of the sequence which is derived by such a node. The algorithm then goes to a loop of k. In each loop, the best node is popped off from the agenda and is stored in a set r. The algorithm adds alternative candidate nodes (or sequences) to the agenda via a double nested loop. The idea is that, when an optimal node (or sequence) is popped off, we have to push to the agenda all nodes (sequences) which are slightly worse than the just popped one. The interpretation of slightly worse is to replace one edge from the popped node (sequence). The slightly worse sequences can be found by the exact heuristic derived from the first Viterbi forward pass.

Figure 3 shows an example of the push operations for a lattice of $T = 4$, $Y = 4$. Suppose an optimal node 2:B (in red, standing for node B at position 2, representing the sequence of 0:A 1:D 2:B 3:C) is popped off, new nodes of 1:A, 1:B, 1:C and 0:B, 0:C and 0:D are pushed to the agenda according to the double nested for loop in Algorithm 3. Each of the pushed nodes represents a sequence, for example, node 1:B represents a sequence which consists of three parts: Viterb sequence from start to 1:B (0:C 1:B), 2:B and forward link of 2:B (3:C in this case). All of these pushed nodes (sequences) are served as candidates for the next agenda pop operation.

The algorithm terminates the loop once it has optimal k nodes. The k-best sequences can be derived by the k optimal nodes. This algorithm has

[6] They did not provide k-best solutions.

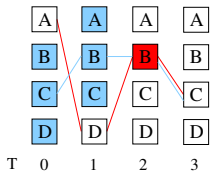

Figure 3: Alternative nodes push after popping an optimal node.

computation complexity of $TL^2 + TL$ for both best and worst cases, with the first term accounting for Viterbi forward pass and the second term accounting for A* backward process. The bottleneck is thus at the Viterbi forward pass.

Algorithm 3 K-Best Viterbi A* algorithm

```
forward()
push L best nodes to agenda q
c = 0
r = {}
while c < K do
    Node n = q.pop()
    r = r ∪ n
    for i = n.t − 1; i ≥ 0; i − − do
        for j = 0; j < L; j + + do
            if j! = n.backlink.y then
                create new node s at position i and label j
                s.forwardlink = n
                q.push(s)
            end if
        end for
        n = n.backlink
    end for
    c + +
end while
return K best sequences derived by r
```

4 Proposed Algorithms

In this section, we propose A* based sequential decoding algorithms that can efficiently handle datasets with a large number of labels. In particular, we first propose the A* and the iterative A* decoding algorithm for 1-best sequential decoding. We then extend the 1-best A* algorithm to a k-best A* decoding algorithm. We finally apply the iterative process to the Viterbi A* algorithm, resulting in the *iterative Viterbi A** decoding algorithm.

4.1 1-Best A*

A*(Hart et al., 1968; Russell and Norvig, 1995), as a classic search algorithm, has been successfully applied in syntactic parsing (Klein and Manning, 2003; Pauls and Klein, 2009). The general idea of A* is to consider labels y_t which are likely to result in the best sequence using a score f as follows.

$$f(y) = g(y) + h(y), \tag{8}$$

where $g(y)$ is the score from start to the current node and $h(y)$ is a heuristic which estimates the score from the current node to the target. A* uses an agenda (based on the f score) to decide which nodes are to be processed next. If the heuristic satisfies the condition $h(y_{t-1}) \geq e(y_{t-1}, y_t) + h(y_t)$, then h is called *monotone* or *admissible*. In such a case, A* is guaranteed to find the best sequence. We start with the naive (but admissible) heuristic as follows

$$h(y_t) = \sum_{i=t+1}^{T-1} (\max n(y_i) + \max e(y_{i-1}, y_i)). \tag{9}$$

That is, the heuristic of node y_t to the end is the sum of max edge scores between any two positions and max node scores per position. Similar to (Pauls and Klein, 2009) we explore the heuristic in different coarse levels. We apply the Viterbi backward pass to different degenerate lattices and use the Viterbi backward scores as different heuristics. Different degenerate lattices are generated from different iterations of Algorithm 1: The m-th iteration corresponds to a lattice of $(2^m + 1) * T$ nodes. A larger m indicates a more accurate heuristic, which results in a more efficient A* search (fewer nodes being processed). However, this efficiency comes with the price that such an accurate heuristic requires more computation time in the Viterbi backward pass. In our experiments, we try the naive heuristic and the following values of m: 0, 3, 6 and 9.

In the best case, A* expands one node per position, and each expansion results in the push of all nodes at next position to the agenda. The search is similar to the beam search with beam size being 1. The complexity is thus TL. In the worst case, A* expands every node per position, and each expansion results in the push of all nodes at next position to the agenda. The complexity thus becomes TL^2.

4.2 1-Best Iterative A*

The iterative process as described in the iterative Viterbi decoding can be used to boost A* algorithm, resulting in the *iterative A** algorithm. For simplicity, we only make use of the naive heuristic in Equation (9) in the iterative A* algorithm. We initialize the lattice with one active label and one degenerate label at each position (see Figure 1 (b)). We then run A* algorithm on the degenerate lattice and get the best sequence. If the sequence is active we return it. Otherwise we expand the lattice in each iteration until we find the best active sequence. Similar to iterative Viterbi algorithm, iterative A* has the complexity of T and TL^2 for the best and worst cases respectively.

4.3 K-Best A*

The extension from 1-best A* to k-best A* is again due to the memorization of k-best labels per node.

Table 1: Best case and worst case computational complexity of various decoding algorithms.

	1-best decoding		K-best decoding	
	best case	worst case	best case	worst case
beam	TL	TL	KTL	KTL
Viterbi	TL^2	TL^2	KTL^2	KTL^2
iterative Viterbi	T	TL^2	N/A	N/A
Carpediem	$TL \log L$	TL^2	N/A	N/A
A*	TL	TL^2	KTL	KTL^2
iterative A*	T	TL^2	N/A	N/A
Viterbi A*	N/A	N/A	$TL^2 + KTL$	$TL^2 + KTL$
iterative Viterbi A*	N/A	N/A	$T + KT$	$TL^2 + KTL$

We use either the naive heuristic (Equation (9)) or different coarse level heuristics by setting m to be 0, 3, 6 or 9 (see Section 4.1). The first k nodes which are popped off the agenda can be used to back track the k-best sequences. The k-best A* algorithm has the computational complexity of KTL and KTL^2 for best and worst cases respectively.

4.4 K-Best Iterative Viterbi A*

We now present the k-best iterative Viterbi A* algorithm (see Algorithm 4) which applies the iterative process to k-best Viterbi A* algorithm. The major difference between 1-best iterative Viterbi A* algorithm (Algorithm 1) and this algorithm is that the latter calls the k-best Vitebi A* (Algorithm 3) after the best sequence is found. If the k-best sequences are all active, we terminate the algorithm and return the k-best sequences. If we cannot find either the best active sequence or the k-best active sequences, we expand the lattice to continue the search in the next iteration.

As in the iterative Viterbi algorithm (see Section 3.2), nodes are pruned at each position in forward or backward passes. Efficient pruning contributes significantly to speeding up decoding. Therefore, to have a tighter (higher) lower bound lb is important. We initialize the lower bound lb with the k-th best score from beam search (with beam size being k) at line 1. Note that the beam search is performed on the original lattice which consists of L active labels per position. The beam search time is negligible compared to the total decoding time. At line 16, we update lb as follows. We enumerate the best active sequences backtracked by the nodes at position $T-1$. If the current lb is less than the k-th active sequence score, we update the lb with the k-th active sequence score (we do not update lb if there are less than k active sequences). At line 19, we use the sequences returned from Viterbi A* algorithm to update the lb in the same manner. To enable this update, we request the Viterbi A* algorithm to return k', $k' > k$, sequences (line 10). A larger number of k' results in a higher chance to find the k-th active sequence, which in turn offers a tighter (higher) lb, but it comes with the expense of additional time (the backward A* process takes $O(TL)$ time to return one more sequence). In experiments, we found the lb updates on line 1 and line 16 are essential for fast decoding. The updating of lb using Viterbi A* sequences (line 19) can boost the decoding speed further. We experimented with different k' values ($k' = nk$, where n is an integer) and selected $k' = 2k$ which results in the largest decoding speed boost.

Algorithm 4 K-Best iterative Viterbi A* algorithm

```
1: lb = k-th best (original lattice)
2: init lattice
3: for i = 0; ; i++ do
4:     if i%2 == 0 then
5:         y = forward()
6:     else
7:         y = backward()
8:     end if
9:     if y consists of active labels only then
10:        ys= k-best Viterbi A* (Algorithm 3)
11:        if ys consists of active sequences only then
12:            return ys
13:        end if
14:    end if
15:    if lb < k-th best(lattice) then
16:        lb = k-th best(lattice)
17:    end if
18:    if lb < k-th best(ys) then
19:        lb = k-th best(ys)
20:    end if
21:    expand lattice
22: end for
```

5 Experiments

We compare aforementioned 1-best and k-best sequential decoding algorithms using five datasets in this section.

5.1 Experimental setting

We apply 1-best and k-best sequential decoding algorithms to five NLP tagging tasks: Penn TreeBank (PTB) POS tagging, CoNLL2000 joint POS tagging and chunking, CoNLL 2003 joint POS tagging, chunking and named entity tagging, HPSG supertagging (Matsuzaki et al., 2007) and a search query named entity recognition (NER) dataset. We used

sections 02-21 of PTB for training and section 23 for testing in POS task. As in (Kaji et al., 2010), we combine the POS tags and chunk tags to form joint tags for CoNLL 2000 dataset, e.g., NN|B-NP. Similarly we combine the POS tags, chunk tags, and named entity tags to form joint tags for CoNLL 2003 dataset, e.g., PRP$|I-NP|O. Note that by such tag joining, we are able to offer different tag decodings (for example, chunking and named entity tagging) simultaneously. This indeed is one of the effective approaches for joint tag decoding problems. The search query NER dataset is an in-house annotated dataset which assigns semantic labels, such as *product*, *business* tags to web search queries.

Table 2 shows the training and test sets size (sentence #), the average token length of test dataset and the label size for the five datasets. POS and supertag datasets assign tags to tokens while CoNLL 2000 , CoNLL 2003 and search query datasets assign tags to phrases. We use the standard BIO encoding for CoNLL 2000, CoNLL 2003 and search query datasets.

Table 2: Training and test datasets size, average token length of test set and label size for five datasets.

	training #	test #	token length	label size
POS	39831	2415	23	45
CoNLL2000	8936	2012	23	319
CoNLL2003	14987	3684	12	443
Supertag	37806	2291	22	2602
search query	79569	6867	3	323

Due to the long CRF training time (days to weeks even for stochastic gradient descent training) for these large label size datasets, we choose the perceptron algorithm for training. The models are averaged over 10 iterations (Collins, 2002). The training time takes minutes to hours for all datasets. We note that the selection of training algorithm does not affect the decoding process: the decoding is identical for both CRF and perceptron training algorithms. We use the common features which are adopted in previous studies, for example (Sha and Periera, 2003). In particular, we use the unigrams of the current and its neighboring words, word bigrams, prefixes and suffixes of the current word, capitalization, all-number, punctuation, and tag bigrams for POS, CoNLL2000 and CoNLL 2003 datasets. For supertag dataset, we use the same features for the word inputs, and the unigrams and bigrams for gold POS inputs. For search query dataset, we use the same features plus gazetteer based features.

5.2 Results

We report the token accuracy for all datasets to facilitate comparison to previous work. They are 97.00, 94.70, 95.80, 90.60 and 88.60 for POS, CoNLL 2000, CoNLL 2003, supertag, and search query respectively. We note that all decoding algorithms as listed in Section 3 and Section 4 are exact. That is, they produce exactly the same accuracy. The accuracy we get for the first four tasks is comparable to the state-of-the-art. We do not have a baseline to compare with for the last dataset as it is not publicly available[7]. Higher accuracy may be achieved if more task specific features are introduced on top of the standard features. As this paper is more concerned with the decoding speed, the feature engineering is beyond the scope of this paper.

Table 3 shows how many iterations in average are required for iterative Viterbi and iterative Viterbi A* algorithms. Although the max iteration size is bounded to $\lceil \log_2 L \rceil$ for each position (for example, 9 for CoNLL 2003 dataset), the total iteration number for the whole lattice may be greater than $\lceil \log_2 L \rceil$ as different positions may not expand at the same time. Despite the large number of iterations used in iterative based algorithms (especially iterative Viterbi A* algorithm), the algorithms are still very efficient (see below).

Table 3: Iteration numbers of iterative Viterbi and iterative Viterbi A* algorithms for five datasets.

	POS	CoNLL2000	CoNLL2003	Supertag	search query
iter Viter	6.32	8.76	9.18	10.63	6.71
iter Viter A*	14.42	16.40	15.41	18.62	9.48

Table 4 and 5 show the decoding speed (sentences per second) of 1-best and 5-best decoding algorithms respectively. The proposed decoding algorithms and the largest decoding speeds across different decoding algorithms (other than beam) are highlighted in bold. We exclude the time for feature extraction in computing the speed. The beam search decoding is also shown as a baseline. We note that beam decoding is the only approximate decoding algorithm in this table. All other decoding algorithms produce exactly the same accuracy, which is usually much better than the accuracy of beam decoding.

For 1-best decoding, iterative Viterbi always outperforms other ones. A* with a proper heuristic denoted as *A* (best)*, that is, the best A* using naive heuristic or the values of m being 0, 3, 6 or 9 (see Section 4.1), can be the second best choice (except for the POS task), although the gap between iterative Viterbi and A* is significant. For example, for CoNLL 2003 dataset, the former can decode 2239 sentences per second while the latter only decodes 225 sentences per second. The iterative process successfully boosts the decoding speed of iterative Viterbi compared to Viterbi, but it slows down the decoding speed of iterative A* compared

[7]The lower accuracy is due to the dynamic nature of queries: many of test query tokens are unseen in the training set.

to A*(best). This is because in the Viterbi case, the iterative process has a node pruning procedure, while it does not have such pruning in A*(best) algorithm. Take CoNLL 2003 data as an example, the removal of the pruning slows down the 1-best iterative Viterbi decoding from 2239 to 604 sentences/second. Carpediem algorithm performs poorly in four out of five tasks. This can be explained as follows. The Carpediem implicitly assumes that the node scores are the dominant factors to determine the best sequence. However, this assumption does not hold as the edge scores play an important role.

For 5-best decoding, k-best Viterbi decoding is very slow. A* with a proper heuristic is still slow. For example, it only reaches 11 sentences per second for CoNLL 2003 dataset. The classic Viterbi A* can usually obtain a decent decoding speed, for example, 40 sentences per second for CoNLL 2003 dataset. The only exception is supertag dataset, on which the Viterbi A* decodes 0.1 sentence per second while the A* decodes 3. This indicates the scalability issue of Viterbi A* algorithm for datasets with more than one thousand labels. The proposed iterative Viterbi A* is clearly the winner. It speeds up the Viterbi A* to factors of 4, 7, 360, and 3 for CoNLL 2000, CoNLL 2003, supertag and query search data respectively. The decoding speed of iterative Viterbi A* can even be comparable to that of beam search.

Figure 4 shows k-best decoding algorithms decoding speed with respect to different k values for CoNLL 2003 data . The Viterbi A* and iterative Viterbi A* algorithms are significantly faster than the Viterbi and A*(best) algorithms. Although the iterative Viterbi A* significantly outperforms the Viterbi A* for $k < 30$, the speed of the former converges to the latter when k becomes 90 or larger. This is expected as the k-best sequences span over the whole lattice: the earlier iteration in iterative Viterbi A* algorithm cannot provide the k-best sequences using the degenerate lattice. The overhead of multiple iterations slows down the decoding speed compared to the Viterbi A* algorithm.

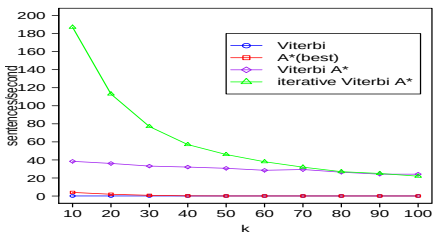

Figure 4: Decoding speed of k-best decoding algorithms for various k for CoNLL 2003 dataset.

6 Related work

The Viterbi algorithm is the only exact algorithm widely adopted in the NLP applications. Esposito and Radicioni (2009) proposed an algorithm which opens necessary nodes in a lattice in searching the best sequence. The staggered decoding (Kaji et al., 2010) forms the basis for our work on iterative based decoding algorithms. Apart from the exact decoding, approximate decoding algorithms such as beam search are also related to our work. Tsuruoka and Tsujii (2005) proposed easiest-first deterministic decoding. Siddiqi and Moore (2005) presented the parameter tying approach for fast inference in HMMs. A similar idea was applied to CRFs as well (Cohn, 2006; Jeong, 2009). We note that the exact algorithm always guarantees the optimality which cannot be attained in approximate algorithms.

In terms of k-best parsing, Huang and Chiang (2005) proposed an efficient algorithm which is similar to the k-best Viterbi A* algorithm presented in this paper. Pauls and Klein (2009) proposed an algorithm which replaces the Viterbi forward pass with an A* search. Their algorithm optimizes the Viterbi pass, while the proposed iterative Viterbi A* algorithm optimizes both Viterbi and A* passes.

This paper is also related to the coarse to fine PCFG parsing (Charniak et al., 2006) as the degenerate labels can be treated as coarse levels. However, the difference is that the coarse-to-fine parsing is an approximate decoding while ours is exact one. In terms of different coarse levels of heuristic used in A* decoding, this paper is related to the work of hierarchical A* framework (Raphael, 2001; Felzenszwalb et al., 2007). In terms of iterative process, this paper is close to (Burkett et al., 2011) as both exploit the search-and-expand approach.

7 Conclusions

We have presented and evaluated the A* and iterative A* algorithms for 1-best sequential decoding in this paper. In addition, we proposed A* and iterative Viterbi A* algorithm for k-best sequential decoding. K-best Iterative A* algorithm can be several times or orders of magnitude faster than the state-of-the-art k-best decoding algorithm. It makes real-time large-scale tagging applications with thousands of labels feasible.

Acknowledgments

We wish to thank Yusuke Miyao and Nobuhiro Kaji for providing us the HPSG Treebank data. We are grateful for the invaluable comments offered by the anonymous reviewers.

Table 4: Decoding speed (sentences per second) of 1-best decoding algorithms for five datasets.

	POS	CoNLL2000	CoNLL2003	supertag	query search
beam	7252	1381	1650	395	7571
Viterbi	2779	51	41	0.19	443
iterative Viterbi	**5833**	**972**	**2239**	**213**	**6805**
Carpediem	2638	14	20	0.15	243
A* (best)	802	131	225	8	880
iterative A*	1112	84	109	3	501

Table 5: Decoding speed (sentences per second) of 5-best decoding algorithms for five datasets.

	POS	CoNLL2000	CoNLL2003	supertag	query search
beam	2760	461	592	75	4354
Viterbi	19	0.41	0.25	0.12	3.83
A* (best)	205	4	11	3	92
Viterbi A*	**1266**	47	40	0.1	357
iterative Viterbi A*	788	**200**	**295**	**36**	**1025**

References

D. Burkett, D. Hall, and D. Klein. 2011. *Optimal graph search with iterated graph cuts*. Proceedings of AAAI.

E. Charniak, M. Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, and T. Vu. 2006. *Multi-level coarse-to-fine PCFG parsing*. Proceedings of NAACL.

T. Cohn. 2006. *Efficient inference in large conditional random fields*. Proceedings of ECML.

M. Collins. 2002. *Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms*. Proceedings of EMNLP.

R. Esposito and D. P. Radicioni. 2009. *Carpediem: Optimizing the Viterbi Algorithm and Applications to Supervised Sequential Learning*. Journal of Machine Learning Research.

P. Felzenszwalb and D. McAllester. 2007. *The generalized A* architecture*. Journal of Artificial Intelligence Research.

P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics.

L. Huang and D. Chiang. 2005. *Better k-best parsing*. Proceedings of the International Workshops on Parsing Technologies (IWPT).

M. Jeong, C. Y. Lin, and G. G. Lee. 2009. *Efficient inference of CRFs for large-scale natural language data*. Proceedings of ACL-IJCNLP Short Papers.

N. Kaji, Y. Fujiwara, N. Yoshinaga, and M. Kitsuregawa. 2010. *Efficient Staggered Decoding for Sequence Labeling*. Proceedings of ACL.

D. Klein and C. Manning. 2003. *A* parsing: Fast exact Viterbi parse selection*. Proceedings of ACL.

J. Lafferty, A. McCallum, and F. Pereira. 2001. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*. Proceedings of ICML.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2007. *Efficient HPSG parsing with supertagging and CFG-filtering*. Proceedings of IJCAI.

A. Pauls and D. Klein. 2009. *K-Best A* Parsing*. Proceedings of ACL.

L. R. Rabiner. 1989. *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of The IEEE.

C. Raphael. 2001. *Coarse-to-fine dynamic programming*. IEEE Transactions on Pattern Analysis and Machine Intelligence.

S. Russell and P. Norvig. 1995. *Artificial Intelligence: A Modern Approach*.

F. Sha and F. Pereira. 2003. *Shallow parsing with conditional random fields*. Proceedings of HLT-NAACL.

S. M. Siddiqi and A. Moore. 2005. *Fast inference and learning in large-state-space HMMs*. Proceedings of ICML.

Y. Tsuruoka and J. Tsujii. 2005. *Bidirectional inference with the easiest-first strategy for tagging sequence data*. Proceedings of HLT/EMNLP.

A. J. Viterbi. 1967. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*. IEEE Transactions on Information Theory.