An Algorithmic Framework for the Decoding Problem in Statistical Machine Translation

Raghavendra Udupa U Tanveer A Faruquie
IBM India Research Lab
Block-1A, IIT, Hauz Khas
New Delhi - 110 016
India
{uraghave, ftanveer}@in.ibm.com

Hemanta K Maji
Dept. of Computer Science
and Engineering, IIT Kanpur
Kanpur - 208 016
India,
hkmaji@iitk.ac.in

Abstract

The decoding problem in Statistical Machine Translation (SMT) is a computationally hard combinatorial optimization problem. In this paper, we propose a new algorithmic framework for solving the decoding problem and demonstrate its utility. In the new algorithmic framework, the decoding problem can be solved both exactly and approximately. The key idea behind the framework is the modeling of the decoding problem as one that involves alternating maximization of two relatively simpler subproblems. We show how the subproblems can be solved efficiently and how their solutions can be combined to arrive at a solution for the decoding problem. A family of provably fast decoding algorithms can be derived from the basic techniques underlying the framework and we present a few illustrations. Our first algorithm is a provably linear time search algorithm. We use this algorithm as a subroutine in the other algorithms. We believe that decoding algorithms derived from our framework can be of practical significance.

1 Introduction

Decoding is one of the three fundamental problems in classical SMT (translation model and language model being the other two) as proposed by IBM in the early 1990's (Brown et al., 1993). In the decoding problem we are given the language and translation models and a source language sentence and are asked to find the most probable translation for the sentence. *Decoding* is a discrete optimization problem whose search space is prohibitively large. The challenge is, therefore, in devising a scheme to efficiently search the solution space for the solution.

Decoding is known to belong to a class of computational problems popularly known as NP-hard problems (Knight, 1999). NP-hard problems are known to be computationally hard and have eluded polynomial time algorithms (Garey and Johnson, 1979). The first algorithms for the decoding problem were based on what is known among the speech recognition community as stack-based search (Jelinek, 1969). The original IBM solution to the decoding problem employed a restricted stack-based search (Berger et al., 1996). This idea was further explored by Wang and Waibel (Wang and Waibel, 1997) who developed a faster stack-based search algorithm. In perhaps the first work on the computational complexity of *Decoding*, Kevin Knight showed that the problem is closely related to the more famous *Traveling Salesman problem* (TSP). Independently, Christoph Tillman adapted the Held-Karp dynamic programming algorithm for TSP (Held and Karp, 1962) to *Decoding* (Tillman, 2001). The original Held-Karp algorithm for TSP is an exponential time dynamic programming algorithm and Tillman's adaptation to *Decoding* has a prohibitive complexity of $O\left(l^3m^2 2^m\right) \approx O\left(m^5 2^m\right)$ (where m and l are the lengths of the source and target sentences respectively). Tillman and Ney showed how to improve the complexity of the Held-Karp algorithm for restricted word reordering and gave a $O\left(l^3m^4\right) \approx O\left(m^7\right)$ algorithm for French-English translation (Tillman and Ney, 2000). An optimal decoder based on the well-known A^* heuristic was implemented and benchmarked in (Och et al., 2001). Since optimal solution can not be computed for practical problem instances in a reasonable amount of time, much of recent work has focused on good quality suboptimal solutions. An $O\left(m^6\right)$ greedy search algorithm was developed (Germann et al., 2003) whose complexity was reduced further to $O\left(m^2\right)$ (Germann, 2003).

In this paper, we propose an algorithmic framework for solving the decoding problem and show that several efficient decoding algorithms can be derived from the techniques developed in the framework. We model the search problem

as an alternating search problem. The search, therefore, alternates between two subproblems, both of which are much easier to solve in practice. By breaking the decoding problem into two simpler search problems, we are able to provide handles for solving the problem efficiently. The solutions of the subproblems can be combined easily to arrive at a solution for the original problem. The first subproblem fixes an alignment and seeks the best translation with that alignment. Starting with an initial alignment between the source sentence and its translation, the second subproblem asks for an improved alignment. We show that both of these problems are easy to solve and provide efficient solutions for them. In an iterative search for a local optimal solution, we alternate between the two algorithms and refine our solution.

The algorithmic framework provides handles for solving the decoding problem at several levels of complexity. At one extreme, the framework yields an algorithm for solving the decoding problem optimally. At the other extreme, it yields a provably linear time algorithm for finding suboptimal solutions to the problem. We show that the algorithmic handles provided by our framework can be employed to develop a very fast decoding algorithm which finds good quality translations. Our fast suboptimal search algorithms can translate sentences that are 50 words long in about 5 seconds on a simple computing facility.

The rest of the paper is devoted to the development and discussion of our framework. We start with a mathematical formulation of the decoding problem (Section 2). We then develop the alternating search paradigm and use it to develop several decoding algorithms (Section 3). Next, we demonstrate the practical utility of our algorithms with the help of results from our initial experiments (Section 5).

2 Decoding

The decoding problem in SMT is one of finding the most probable translation $\hat{\mathbf{e}}$ in the target language of a given source language sentence $\mathbf{f}$ in accordance with the Fundamental Equation of SMT (Brown et al., 1993):

$$\hat{\mathbf{e}} = argmax_{\mathbf{e}} \quad Pr(\mathbf{f}|\mathbf{e})Pr(\mathbf{e}). \tag{1}$$

In the remainder of this paper we will refer to the search problem specified by Equation 1 as **STRICT_DECODING**.

Rewriting the translation model $Pr(\mathbf{f}|\mathbf{e})$ as $\sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$, where $\mathbf{a}$ denotes an alignment between the source sentence and the target sentence, the problem can be restated as:

$$\hat{\mathbf{e}} = argmax_{\mathbf{e}} \quad \sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})Pr(\mathbf{e}). \tag{2}$$

Even when the translation model $Pr(\mathbf{f}|\mathbf{e})$ is as simple as IBM Model 1 and the language model $Pr(\mathbf{e})$ is a bigram language model, the decoding problem is NP-hard (Knight, 1999). Unless P = NP, there is no hope of an efficient algorithm for the decoding problem. Since the Fundamental Equation of SMT does not yield an easy handle to design a solution (exact or even an approximate one) for the problem, most researchers have instead worked on solving the following relatively simpler problem (Germann et al., 2003):

$$(\hat{\mathbf{e}}, \hat{\mathbf{a}}) = argmax_{(\mathbf{e},\mathbf{a})} \quad Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})Pr(\mathbf{e}). \tag{3}$$

We call the search problem specified by Equation 3 as **RELAXED_DECODING**. Note that **RELAXED_DECODING** relaxes **STRICT_DECODING** to a joint optimization problem. The search in **RELAXED_DECODING** is for a pair $(\hat{\mathbf{e}}, \hat{\mathbf{a}})$. While **RELAXED_DECODING** is simpler than **STRICT_DECODING**, it is also, unfortunately, NP hard for even IBM Model 1 and Bigram language model. Therefore, all practical solutions to **RELAXED_DECODING** have focused on finding suboptimal solutions. The challenge is in devising fast search strategies to find good suboptimal solutions. Table 1 lists the combinatorial optimization problems in the domain of decoding.

In the remainder of the paper, m and l denote the length of the source language sentence and its translation respectively.

3 Framework for Decoding

We begin with a couple of useful observations about the decoding problem. Although deceptively simple, these observations are very crucial for developing our framework. They are the source for algorithmic handles for breaking the decoding problem into two relatively easier search problems. The first of these observations concerns with solving the problem when we know in advance the mapping between the source and target sentences. This leads to the development of an extremely simple algorithm for decoding when the alignment is known (or

Problem	Search
STRICT_DECODING	$\hat{\mathbf{e}} = argmax_{\mathbf{e}} Pr(\mathbf{f}\|\mathbf{e})Pr(\mathbf{e})$
RELAXED_DECODING	$(\hat{\mathbf{e}}, \hat{\mathbf{a}}) = argmax_{(\mathbf{e},\mathbf{a})} Pr(\mathbf{f}, \mathbf{a}\|\mathbf{e})Pr(\mathbf{e})$
FIXED_ALIGNMENT_DECODING	$\hat{\mathbf{e}} = argmax_{\mathbf{e}} Pr(\mathbf{f}, \tilde{\mathbf{a}}\|\mathbf{e})Pr(\mathbf{e})$
VITERBI_ALIGNMENT	$\hat{\mathbf{a}} = argmax_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a}\|\tilde{\mathbf{e}})$

Table 1: Combinatorial Search Problems in Decoding

can be guessed). Our second observation is on finding a better alignment between the source and target sentences starting with an initial (possibly suboptimal) alignment. The insight provided by the two observations are employed in building a powerful algorithmic framework.

3.1 Handles for attacking the Decoding Problem

Our goal is to arrive at algorithmic handles for attacking RELAXED_DECODING. In this section, we make couple of useful observations and develop algorithmic handles from the insight provided by them. The first of the two observations is:

Observation 1 *For a given target length l and a given alignment $\tilde{\mathbf{a}}$ that maps source words to target positions, it is easy to compute the optimal target sentence $\hat{\mathbf{e}}$.*

$$\hat{\mathbf{e}} = argmax_{\mathbf{e}} \;\; Pr(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})Pr(\mathbf{e}). \qquad (4)$$

Let us call the search problem specified by Equation 4 as FIXED_ALIGNMENT_DECODING. What Observation 1 is saying is that once the target sentence length and the source to target mapping is fixed, the optimal target sentence (with the specified target length and alignment) can be computed efficiently. As we will show later, the optimal solution for FIXED_ALIGNMENT_DECODING can be computed in $O(m)$ time for IBM models 1-5 using dynamic programming. As we can always guess an alignment (as is the case with many decoding algorithms in the literature), the above observation provides an algorithmic handle for finding suboptimal solutions for RELAXED_DECODING.

Our second observation is on computing the optimal alignment between the source sentence and the target sentence.

Observation 2 *For a given target sentence $\tilde{\mathbf{e}}$, it is easy to compute the optimal alignment $\hat{\mathbf{a}}$ that maps the source words to the target words.*

$$\hat{\mathbf{a}} = argmax_{\mathbf{a}} \;\; Pr(\mathbf{f}, \mathbf{a}|\tilde{\mathbf{e}}). \qquad (5)$$

It is easy to determine the optimal (Viterbi) alignment between the source sentence and its translation. In fact, for IBM models 1 and 2, the Viterbi alignment can be computed using a straight forward algorithm in $O(ml)$ time. For higher models, an approximate Viterbi alignment can be computed iteratively by an iterative procedure called local search. In each iteration of local search, we look in the neighborhood of the current best alignment for a better alignment (Brown et al., 1993). The first iteration can start with any arbitrary alignment (say the Viterbi alignment of Model 2). It is possible to implement one iteration of local search in $O(ml)$ time. Typically, the number of iterations is bounded in practice by $O(m)$, and therefore, local search takes $O(m^2 l)$ time.

Our framework is not strictly dependent on the computation of an optimal alignment. Any alignment which is better than the current alignment is good enough for it to work. It is straight forward to find one such alignment using restricted swaps and moves in $O(m)$ time. In the remainder of this paper, we use the term *Viterbi* to denote any linear time algorithm for computing an improved alignment between the source sentence and its translation.

3.2 Illustrative Algorithms

In this section, we show how the handles provided by the above two observations can be employed to solve RELAXED_DECODING. The two handles are in some sense complementary to each other. When the alignment is known, we can efficiently determine the optimal translation with that alignment. On the other hand, when the translation is known, we can efficiently determine a better alignment. Therefore, we can use one to improve the other. We begin with the following simple linear time decoding algorithm which is based on the first observation.

Algorithm NaiveDecode

Input: Source language sentence $\mathbf{f}$ of length $m > 0$.

Optional Inputs: Target sentence length l, alignment $\tilde{\mathbf{a}}$ between the source words and target positions.

Output: Target language sentence $\hat{\mathbf{e}}$ of length l.

1. If l is not specified, let $l = m$.
2. If an alignment is not specified, guess some alignment $\tilde{\mathbf{a}}$.
3. Compute the optimal translation $\hat{\mathbf{e}}$ by solving FIXED_ALIGNMENT_DECODING, i.e., $\hat{\mathbf{e}} = argmax_{\mathbf{e}} \ Pr(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})Pr(\mathbf{e})$.
4. return $\hat{\mathbf{e}}$.

When the length of the translation is not specified, *NaiveDecode* assumes that the translation is of the same length as the source sentence. If an alignment that maps the source words to target positions is not specified, the algorithm guesses an alignment $\tilde{\mathbf{a}}$ ($\tilde{\mathbf{a}}$ can be the trivial alignment that maps the source word f_j to target position j, that is, $\tilde{\mathbf{a}}_j = j$, or can be guessed more intelligently). It then computes the optimal translation for the source sentence $\mathbf{f}$, with the length of the target sentence and the alignment between the source and the target sentences kept fixed to l and $\tilde{\mathbf{a}}$ respectively, by maximizing $Pr(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})Pr(\mathbf{e})$. As FIXED_ALIGNMENT_DECODING can be solved in $O(m)$ time, *NaiveDecode* takes only $O(m)$ time.

The value of *NaiveDecode* lies not in itself per se, but in its instrumental role in designing more superior algorithms. The power of *NaiveDecode* can be demonstrated with the following optimal algorithm for RELAXED_DECODING.

Algorithm NaiveOptimalDecode

Input: Source language sentence $\mathbf{f}$ of length $m > 0$.

Output: Target language sentence $\hat{\mathbf{e}}$ of length $l, \frac{m}{2} \leq l \leq 2m$.

1. Let $\hat{\mathbf{e}} = null$ and $\hat{\mathbf{a}} = null$.
2. For each $l = \frac{m}{2}, \ldots, 2m$ do
3. For each alignment $\mathbf{a}$ between the source words and the target positions do
   (a) Let $\mathbf{e} = NaiveDecode(\mathbf{f}, l, \mathbf{a})$.
   (b) If $Pr(\mathbf{f}, \mathbf{e}, \mathbf{a}) > Pr(\mathbf{f}, \hat{\mathbf{e}}, \hat{\mathbf{a}})$ then
      i. $\hat{\mathbf{e}} = \mathbf{e}$
      ii. $\hat{\mathbf{a}} = \mathbf{a}$.
4. return $(\hat{\mathbf{e}}, \hat{\mathbf{a}})$.

NaiveOptimalDecode considers various target lengths and all possible alignments between the source words and the target positions. For each target length l and alignment $\mathbf{a}$ it employs *NaiveDecode* to find the best solution. There are $(l+1)^m$ candidate alignments for a target length l and $O(m)$ candidate target lengths. Therefore, *NaiveOptimalDecode* explores $\Theta(m(l+1)^m)$ alignments. For each of these candidate alignments, it makes a call to *NaiveDecode*. The time complexity of *NaiveOptimalDecode* is, therefore, $O(m^2(l+1)^m)$. Although an exponential time algorithm, it can compute the optimal solution for RELAXED_DECODING.

With *NaiveDecode* and *NaiveOptimalDecode* we have demonstrated the power of the algorithmic handle provided by Observation 1. It is important to note that these two algorithms are at the two extremities of the spectrum. *NaiveDecode* is a linear time decoding algorithm that computes a suboptimal solution for RELAXED_DECODING while *NaiveOptimalDecode* is an exponential time algorithm that computes the optimal solution. What we want are algorithms that are close to *NaiveDecode* in complexity and to *NaiveOptimalDecode* in quality. It is possible to reduce the complexity of *NaiveOptimalDecode* significantly by carefully reducing the number of alignments that are examined. Instead of examining all $\Theta(m(l+1)^m)$ alignments, if we examine only a small number, say $g(m)$, alignments in *NaiveOptimalDecode*, we can find a solution in $O(mg(m))$ time. In the next section, we show how to restrict the search to only a small number of promising alignments.

3.3 Alternating Maximization

We now show how to use the two algorithmic handles to come up with a fast search paradigm. We alternate between searching the best translation given an alignment and searching the best alignment given a translation. Since the two subproblems are complementary, they can be used to improve the solution computed by one another by alternating between the two problems.

Algorithm AlternatingSearch

Input: Source language sentence $\mathbf{f}$ of length $m > 0$.

Output: Target language sentence $\mathbf{e^{(o)}}$ of length l $(m/2 \leq l \leq 2m)$.

1. Let $\mathbf{e^{(o)}} = null$ and $\mathbf{a^{(o)}} = null$.
2. For each $l = m/2, \ldots, 2m$ do
   (a) Let $\mathbf{e} = null$ and $\mathbf{a} = null$.
   (b) While there is improvement in solution do
      i. Let $\mathbf{e} = NaiveDecode\,(\mathbf{f}, l, \mathbf{a})$.
      ii. Let $\hat{\mathbf{a}} = Viterbi\,(\mathbf{f}, \mathbf{e})$.
   (c) If $Pr\,(\mathbf{f}, \mathbf{e}, \mathbf{a}) > Pr\left(\mathbf{f}, \mathbf{e^{(o)}}, \mathbf{a^{(o)}}\right)$ then
      i. $\mathbf{e^{(o)}} = \mathbf{e}$
      ii. $\mathbf{a^{(o)}} = \mathbf{a}$.
3. return $\mathbf{e^{(o)}}$.

AlternatingSearch searches for a good translation by varying the length of the target sentence. For a sentence length l, the algorithm finds a translation of length l and then iteratively improves the translation. In each iteration it solves two subproblems: FIXED_ALIGNMENT_DECODING and VITERBI_ALIGNMENT. The input to each iteration are the source sentence $\mathbf{f}$, the target sentence length l, and an alignment $\mathbf{a}$ between the source and target sentences. So, *AlternatingSearch* finds a better translation $\mathbf{e}$ for $\mathbf{f}$ by solving FIXED_ALIGNMENT_DECODING. For this purpose it employs *NaiveDecode*. Having computed $\mathbf{e}$, the algorithm computes a better alignment $(\hat{\mathbf{a}})$ between $\mathbf{e}$ and $\mathbf{f}$ by solving VITERBI_ALIGNMENT using *Viterbi* algorithm. The new alignment thus found is used by the algorithm in the subsequent iteration. At the end of each iteration the algorithm checks whether it has made progress. The algorithm returns the best translation of the source $\mathbf{f}$ across a range of target sentence lengths.

The analysis of *AlternatingSearch* is complicated by the fact that the number of iterations (see step 2.b) depends on the input. It is reasonable to assume that the length of the source sentence (m) is an upper bound on the number of iterations. In practice, however, the number of iterations is typically $O\,(1)$. There are 3m/2 candidate sentence lengths for the translation (l varies from $m/2$ to $2m$) and both *NaiveDecode* and *Viterbi* are $O\,(m)$. Therefore, the time complexity of *AlternatingSearch* is $O\left(m^2\right)$.

4 A Linear Time Algorithm for FIXED_ALIGNMENT_DECODING

A key component of all our algorithms is a linear time algorithm for the problem FIXED_ALIGNMENT_DECODING. Recall that in FIXED_ALIGNMENT_DECODING, we are given the target length l and a mapping $\tilde{a}$ from source words to target positions. The goal is then to find the optimal translation with $\tilde{a}$ as the alignment. In this section, we give a dynamic programming based solution to this problem. Our solution is based on a new formulation of IBM translation models. We begin our discussion with a few technical definitions.

Alignment $\tilde{a}$ maps each of the source words $f_j, j = 1, \ldots, m$ to a target position in the range $[0, \ldots, l]$. Define a mapping ψ from $[0, \ldots, l]$ to subsets of $\{1, \ldots, m\}$ as follows:

$$\psi(i) = \{j : j \in \{1, \ldots, m\} \wedge \tilde{a}_j = i\} \vee i = 0, \ldots, l.$$

$\psi(i)$ is the set of source positions which are mapped to the target location i by the alignment $\tilde{a}$ and the fertility of the target position i is $\phi_i = |\psi(i)|$.

We can rewrite each of the IBM models $Pr\,(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})$ as follows:

$$Pr\,(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e}) = \xi \prod_{i=1}^{l} \mathcal{T}_i \mathcal{D}_i \mathcal{N}_i.$$

Table 2 shows the breaking of $Pr\,(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})$ into the constituents $\mathcal{T}_i, \mathcal{D}_i$ and $\mathcal{N}_i$. As a consequence, we can write $Pr\,(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})\,Pr\,(\mathbf{e})$ as:

$$Pr\,(\mathbf{f}, \tilde{\mathbf{a}}|\mathbf{e})\,Pr\,(\mathbf{e}) = \xi\lambda \prod_{i=1}^{l} \mathcal{T}_i \mathcal{D}_i \mathcal{N}_i \mathcal{L}_i$$

where $\mathcal{L}_i = trigram(e_i|e_{i-2}, e_{i-1})$ and λ is the trigram probability of the boundary word.

The above reformulation of the optimization function of the decoding problem allows us to employ Dynamic Programming for solving FIXED_ALIGNMENT_DECODING efficiently. Note that each word e_i has only a constant number of candidates in the vocabulary. Therefore, the set of words $e_1, \ldots, e_l$ that maximizes the LHS of the above optimization function can be found in $O\,(m)$ time using the standard Dynamic Programming algorithm (Cormen et al., 2001).

5 Experiments and Results

In this section we describe our experimental setup and present the initial results. Our goal

Model	ξ	$\mathcal{T}_i$	$\mathcal{D}_i$	$\mathcal{N}_i$
1	$\frac{\epsilon(m\|l)}{(l+1)^m}$	$\prod_{k\in\psi(i)} t(f_k\|e_i)$	1	1
2	$\epsilon(m\|l)$	$\prod_{k\in\psi(i)} t(f_k\|e_i)$	$\prod_{k\in\psi(i)} a(i\|k,m,l)$	1
3	$n(\phi_0\|m)p_0^{m-2\phi_0}p_1^{\phi_0}$	$\prod_{k\in\psi(i)} t(f_k\|e_i)$	$\prod_{k\in\psi(i)} d(k\|i,m,l)$	$\phi_i!\ n(\phi_i\|e_i)$

Table 2: $Pr\,(f,\tilde{a}|e)$ for IBM Models

was not only to evaluate the performance of our algorithms on real data, but also to evaluate how easy it is to code the algorithm and whether a straightforward implementation of the algorithm with no parameter tuning can give satisfactory results.

We implemented the algorithms in C++ and conducted the experiments on an IBM RS-6000 dual processor machine with 1 GB of RAM. We built a French-English translation model (IBM Model 3) by training over a corpus of 100 K sentence pairs from the Hansard corpus. The translation direction was from French to English. We built an English language model by training over a corpus consisting of about 800 million words. We divided the test sentences into several classes based on their length. Each length class consisted of 300 test French sentences. We implemented four algorithms -1.1 (*NaiveDecode*), 1.2 (*Alternating Search* with l restricted to m), 2.1 (*NaiveDecode* with l varying from $m/2$ to $2m$) and 2.2 (*Alternating Search*). In order to compare the performance of the algorithms proposed in this paper with a previous decoding algorithm, we also implemented the dynamic programming based algorithm by (Tillman, 2001). For each of the algorithms, we computed the following:

1. Average time taken for translation for each length class.
2. NIST score of the translations for each length class.
3. Average value of the optimization function for the translations for each length class.

The results of the experiments are summarized in Plots 1, 2 and 3. In all the plots, the length class is denoted by the x-axis. 11-20 indicates the class with sentences of length between 11 words to 20 words. 51 indicates the group of sentences with sentence length 51 or more. Plot 1 shows the average time taken by the algorithms for translating the sentences in each length class. Time is shown in seconds on a log scale. Plot 2 shows the NIST score of the translations for each length class while Plot 3 shows the average log score of the translations (-ve log of $Pr\,(\mathbf{f},\mathbf{a}|\mathbf{e})\,Pr\,(\mathbf{e})$) again for each length class.

It can be seen from Plot 1 that all of our algorithms are indeed very fast in practice. They are, in fact, an order faster than the Held-Karp algorithm. Our algorithms are able to translate even long sentences (50+ words) in a few seconds.

Plot 3 shows that the log scores of the translations computed by our algorithms are very close to those computed by the Held-Karp algorithm. Plot 2 compares the NIST scores obtained with each of the algorithm. Among the four algorithms based on our framework, Algorithm 2.2 gives the best NIST scores as expected. Although, the log scores of our algorithms are comparable to those of the Held-Karp algorithm, our NIST scores are lower. It should be noted that the mathematical quantity that our algorithm tries to optimize is the log score. Plot 3 shows that our algorithms are quite good at finding solutions with good scores.

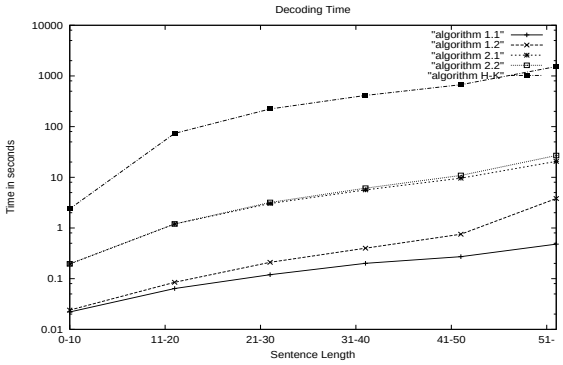

Figure 1: Average decoding time

6 Conclusions

The algorithmic framework developed in this paper is powerful as it yields several decoding algorithms. At one end of the spectrum is a provably linear time algorithm for computing a suboptimal solution and at the other end is an exponential time algorithm for computing

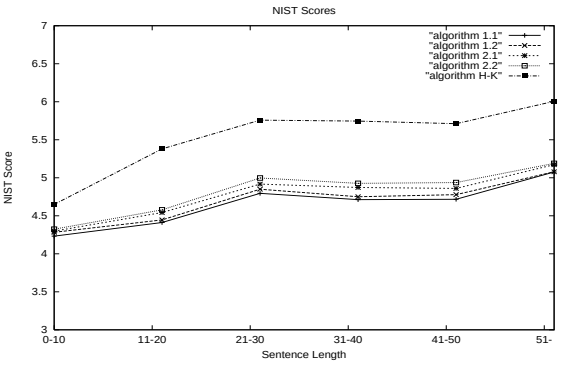

Figure 2: NIST scores

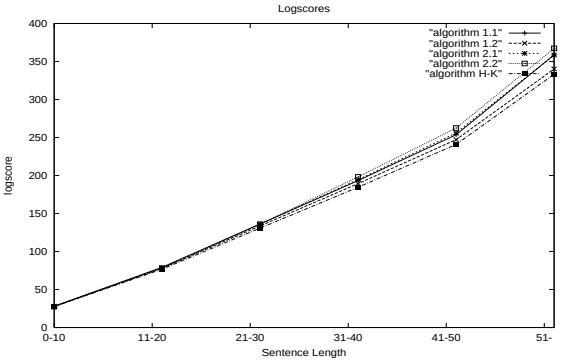

Figure 3: Log score

the optimal solution. We have also shown that alternating maximization can be employed to come up with $O(m^2)$ decoding algorithm. Two questions in this connection are:

1. Is it possible to reduce the complexity of *AlternatingSearch* to $O(m)$?
2. Instead of exploring each alignment separately, is it possible to explore a bunch of alignments in one shot?

Answers to these questions will result in faster and more efficient decoding algorithms.

7 Acknowledgements

We are grateful to Raghu Krishnapuram for his insightful comments on an earlier draft of this paper and Pasumarti Kamesam for his help during the course of this work.

References

A. Berger, P. Brown, S. Della Pietra, V. Della Pietra, A. Kehler, and R. Mercer. 1996. Language translation apparatus and method using context-based translation models. *United States Patent 5,510,981.*

P. Brown, S. Della Pietra, V. Della Pietra, and R. Mercer. 1993. The mathematics of machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. The MIT Press, Cambridge.

M. R. Garey and D. S. Johnson. 1979. W. H. Freeman and Company, New York.

U. Germann, M. Jahr, D. Marcu, and K. Yamada. 2003. Fast decoding and optimal decoding for machine translation. *Artificial Intelligence.*

Ulrich Germann. 2003. Greedy decoding for statistical machine translation in almost linear time. In *Proceedings of HLT-NAACL 2003.* Edmonton, Canada.

M. Held and R. Karp. 1962. A dynamic programming approach to sequencing problems. *J. SIAM*, 10(1):196–210.

F. Jelinek. 1969. A fast sequential decoding algorithm using a stack. *IBM Journal Reseach and Development*, 13:675–685.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4).

F. Och, N. Ueffing, and H. Ney. 2001. An efficient a* search algorithm for statistical machine translation. In *Proceedings of the ACL 2001 Workshop on Data-Driven Methods in Machine Translation*, pages 55–62. Toulouse, France.

C. Tillman and H. Ney. 2000. Word reordering and dp-based search in statistical machine translation. In *Proceedings of the 18th COLING*, pages 850–856. Saarbrucken, Germany.

Christoph Tillman. 2001. Word re-ordering and dynamic programming based search algorithm for statistical machine translation. *Ph.D. Thesis, University of Technology Aachen*, pages 42–45.

R. Udupa and T. Faruquie. 2004. An english-hindi statistical machine translation system. In *Proceedings of the 1st IJCNLP*, pages 626–632. Sanya, Hainan Island, China.

Y. Wang and A. Waibel. 1997. Decoding algorithm in statistical machine translation. In *Proceedings of the 35th ACL*, pages 366–372. Madrid, Spain.