A New Decoding Algorithm for Statistical Machine Translation: Design and Implementation

Tanveer A. Faruquie * Hemanta K. Maji * Raghavendra Udupa U. *

Abstract

We describe a new algorithm for the *Decoding* problem in Statistical Machine Translation. Our algorithm is based on the Alternating Optimization framework and employs dynamic programming. The time complexity of the algorithm is $O\left(m^2\right)$, where m is the length of the sentence to be translated, which is the best among all known algorithms for the problem. As the search space explored by the algorithm is large, we propose two pruning techniques. Empirical results obtained by extensive experimentation on test data show that the new algorithm's runtime grows only linearly with m when either of the pruning techniques is employed. Our algorithm outperforms the best known decoding algorithms and a comparative experimental study shows that our implementation is 10 times faster than the implementation of the *Greedy* decoding algorithm released by [15].

1 Problem Specification.

We study the *Decoding* problem in Statistical Machine Translation (SMT): given a French sentence $\mathbf{f}$ and a probability distribution $Pr\left(\mathbf{e}|\mathbf{f}\right)$, find the most probable English translation of $\mathbf{f}$ [1] [3]:

(1.1)

$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{argmax}\ \ Pr\left(\mathbf{e}|\mathbf{f}\right) = \underset{\mathbf{e}}{argmax}\ \ Pr\left(\mathbf{f}|\mathbf{e}\right) Pr\left(\mathbf{e}\right).$$

Because of the particular structure of the distribution $Pr\left(\mathbf{f}|\mathbf{e}\right)$ employed in SMT, the above problem is recasted in the following form:

$$(\hat{\mathbf{e}}, \hat{\mathbf{a}}) = \underset{\mathbf{e},\mathbf{a}}{argmax}\ \ Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right) Pr\left(\mathbf{e}\right) \tag{1.2}$$

where $\mathbf{a}$ is a many-to-one mapping from the words of the sentence $\mathbf{f}$ to the words of $\mathbf{e}$. In SMT parlance $Pr\left(\mathbf{f}|\mathbf{e}\right)$, $Pr\left(\mathbf{e}\right)$, and $\mathbf{a}$ are known as *Translation Model*, *Language Model*, and *alignment* respectively [3].

Decoding is one of the three fundamental problems in SMT and the only discrete optimization problem of the three [10], [1]. The problem is known to be NP-hard even in the simplest setting [1]. In applications such as speech-to-speech translation and automatic webpage translation, the translation system is expected to have a very high throughput. In other words, the *Decoder* should generate reasonably good translations very quickly. Therefore, our primary goal is to develop a fast decoding algorithm which produces good but suboptimal translations. Another goal is to design the decoding algorithm such that it is easy to implement and requires nearly no tuning for best performance.

Several solutions have been proposed for this problem. The original IBM solution to the decoding problem employed a restricted stack-based search [10] which takes exponential time in the worst case. An adaptation of the Held-Karp dynamic programming based TSP algorithm to the problem runs in $O\left(l^3m^4\right) \approx O\left(m^7\right)$ time (where m and l are the lengths of the sentence and its translation respectively) under certain assumptions [4]. For small sentence lengths, optimal solution to the decoding problem can be found using either the A^* heuristic [7] or integer linear programming [2]. The fastest and arguably the most famous decoding algorithm employs a greedy decoding strategy and finds a suboptimal solution in $O\left(m^6\right)$ time [2] [2]. A more complex greedy decoding algorithm finds a suboptimal solution in $O\left(m^2\right)$ time [6]. An implementation of the greedy algorithm has been provided by its designers on their website [15].

In [14] we described an algorithmic framework for solving the decoding problem. The framework is based on alternating optimization in which the decoding problem is divided into two subproblems each of which can be solved efficiently and combined to refine the solution in an iterative fashion.

In this paper, we develop an $O\left(m^2\right)$ algorithm in the alternating optimization framework (Section 2.3). The key idea is to construct a reasonably large subspace of the search space of the problem and design a computationally efficient search scheme for finding the best solution in the subspace. We show how to construct

*IBM India Research Lab, Block-1A, IIT, Hauz Khas, New Delhi.

[1] Following the custom in SMT literature, we use French and English as the language pair in our discussion. The ideas described in this paper are applicable to any language pair.

[2] The conference version of the paper on greedy decoding received a Best Paper Award at the 39th Annual Meeting of the Association for Computation Linguistics (ACL-2001).

a family of alignments (with $\Theta(4^m)$ alignments) starting with any alignment (Section 3). We employ four *alignment transformation operations* to build a family of alignments from the initial alignment (3.1). We propose a dynamic programming algorithm to find the optimal solution for the decoding problem within the family of alignments thus constructed (Section 3.3). Although the number of alignments in the subspace is exponential in m, the dynamic algorithm is able to compute the optimal solution in $O(m^2)$ time. We extend the algorithm to explore several such families of alignments iteratively (Section 3.4). We look into the engineering aspects of the algorithm and propose two intuitive and simple heuristics to speedup the search (Section 3.5). By caching some of the data used in the computations, we improve the speed further (Section 3.6). Empirical results show that our algorithm is about 10 times faster than the greedy algorithm while computing better solutions (Section 4).

2 The Decoding Problem

2.1 Preliminaries Let $\mathbf{f}$ and $\mathbf{e}$ denote a French sentence and an English sentence respectively. Suppose $\mathbf{f}$ has $m > 0$ words and $\mathbf{e}$ has $l > 0$ words. We can write $\mathbf{f} = f_1 f_2 \dots f_m$ and $\mathbf{e} = e_1 e_2 \dots e_l$, where f_j (e_i) denotes the jth (ith) word in the French (English) sentence. For technical reasons, we prepend the null word e_0 to every English sentence [3].

DEFINITION 2.1. (**Alignment**) *An alignment,* $\mathbf{a}$*, is a mapping which associates each word* $f_j, \quad j = 1, \dots, m$ *in the French sentence (*$\mathbf{f}$*) to some word* $e_{a_j}, \quad a_j \in \{0, \dots, l\}$ *in the English sentence* $\mathbf{e}$*.*

Equivalently, we can speak of $\mathbf{a}$ as a many-to-one mapping from the words of $\mathbf{f}$ to the word positions $0, \dots, l$ in $\mathbf{e}$. We can write $\mathbf{a}$ as $\mathbf{a} = a_1 a_2 \dots a_m$ with the meaning f_j is mapped to e_{a_j}.

Figure 1 shows an alignment $\mathbf{a}$ for the sentence pair $\mathbf{f}, \mathbf{e}$. This particular alignment associates f_1 with e_1 (i.e. $a_1 = 1$) and f_2 with e_0 (i.e. $a_2 = 0$). We note that f_3 and f_4 are mapped to e_2 by $\mathbf{a}$.

DEFINITION 2.2. (**Fertility**) *The fertility of* $e_i, \quad i = 0, \dots, l$ *in an alignment* $\mathbf{a}$ *is the number of words of* $\mathbf{f}$ *mapped to it by* $\mathbf{a}$*. Let* ϕ_i *denote the fertility of* $e_i, \quad i = 0, \dots, l$*.*

In the alignment shown in Figure 1, the fertility of e_2 is 2 as f_3 and f_4 are mapped to it by the alignment while the fertility of e_3 is 0. A word with non-zero fertility is called a *fertile* word and a word with zero fertility is called a *infertile* word. The maximum fertility of an English word is denoted by ϕ_{max} and is typically a small constant.

[3] Null word is necessary to account for French words which are not associated to any of the words in $\mathbf{e}$.

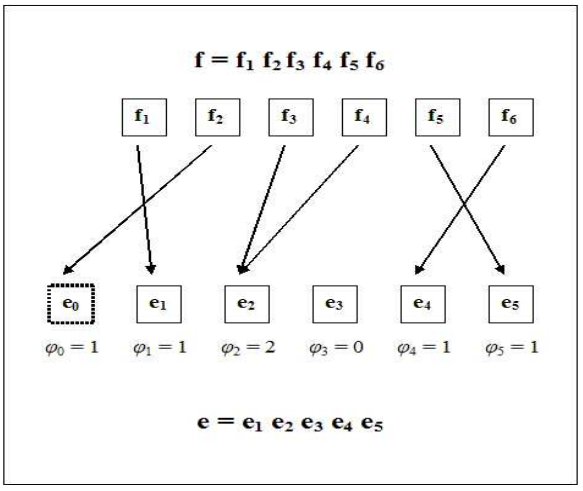

Figure 1: Alignment

Associated with every alignment are a *tableau* and a *permutation*. Tableau is a partition of the words in the sentence $\mathbf{f}$ induced by the alignment and permutation is an ordering of the words in the partition.

DEFINITION 2.3. (**Tableau**) *Let* τ *be a mapping from* $[0, \dots, l]$ *to subsets of* $\{f_1, \dots, f_m\}$ *defined as follows:*

$$\tau_i = \{f_j : j \in \{1, \dots, m\} \ \wedge \ a_j = i\} \ \forall \ i = 0, \dots, l$$

τ_i *is the set of French words which are mapped to the word position* i *in the translation by the alignment.* $\tau_i, \quad i = 0, \dots, l$ *are called the tablets induced by the alignment* $\mathbf{a}$ *and* τ *is called a tableau. The* k*th word in the tablet* τ_i *is denoted by* τ_{ik}*.*

DEFINITION 2.4. (**Permutation**) *Let* π *be a mapping from* $[0, \dots, l]$ *to subsets of* $\{1, \dots, m\}$ *defined as follows:*

$$\pi_i = \{j : j \in \{1, \dots, m\} \ \wedge \ a_j = i\} \ \forall \ i = 0, \dots, l.$$

π_i *is the set of positions which are mapped to position* i *by the alignment* $\mathbf{a}$*. The fertility of* e_i *is* $\phi_i = |\pi_i|$*. We can assume that the positions in the set* π_i *are ordered, i.e.* $\pi_{ik} < \pi_{ik+1}, \quad k = 1, \dots, \phi_i - 1$*. We may further assume that* $\tau_{ik} = f_{\pi_{ik}} \ \forall \ i = 0, \dots, l \ \ \forall \ k = 1, \dots, \phi_i$*.* π *is called a permutation.*

It is easy to see that there is a unique alignment corresponding to a tableau and a permutation.

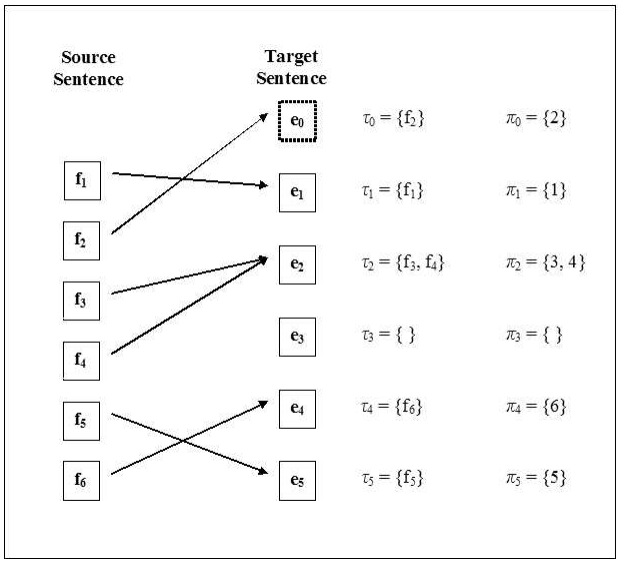

Figure 2: Example of Tableau and Permutation

2.2 Probability Models In SMT, every English sentence $\mathbf{e}$ is a translation of $\mathbf{f}$. Some translations are more likely than others. The probability of $\mathbf{e}$ is $Pr(\mathbf{e}|\mathbf{f})$. In SMT literature, the distribution $Pr(\mathbf{e}|\mathbf{f})$ is replaced by the product $Pr(\mathbf{f}|\mathbf{e})\,Pr(\mathbf{e})$ (by applying Bayes' rule) for technical reasons [3]. Furthermore, a hidden alignment is assumed to exist for each pair $(\mathbf{f}, \mathbf{e})$ with a probability $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ and the translation model $(Pr(\mathbf{f}|\mathbf{e}))$ is expressed as a sum of $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ over all alignments: $Pr(\mathbf{f}|\mathbf{e}) = \sum_{\mathbf{a}} Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$.

SMT techniques model $Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$ and $Pr(\mathbf{e})$ using simpler models which work at the level of words. [3] proposed a set of 5 translation models, commonly known as IBM 1-5. In practice, IBM-4 along with the trigram language model is known to give better translations than other models. Therefore, in the remainder of this paper we describe our decoding algorithm in the context of IBM-4 and trigram language model only, although our methods can be applied to other IBM models as well.

2.2.1 Factorization of Models While IBM 1-5 models can be factorized in many ways, we propose a factorization which is useful in solving the decoding problem efficiently. Our factorization is along the words of the translation:

$$Pr(\mathbf{f}, \mathbf{a}|\mathbf{e}) = \prod_{i=0}^{l} \mathcal{T}_i \mathcal{D}_i \mathcal{N}_i,$$

$$Pr(\mathbf{e}) = \prod_{i=0}^{l} \mathcal{L}_i,$$

and therefore,

$$Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})\, Pr(\mathbf{e}) = \prod_{i=0}^{l} \mathcal{T}_i \mathcal{D}_i \mathcal{N}_i \mathcal{L}_i.$$

Here, the terms $\mathcal{T}_i, \mathcal{D}_i, \mathcal{N}_i$, and $\mathcal{L}_i$ are associated with e_i. The terms $\mathcal{T}_i, \mathcal{D}_i$ and $\mathcal{N}_i$ are determined by the tableau and the permutation induced by the alignment. Only $\mathcal{L}_i$ is Markovian.

IBM-4 employs distributions $t()$ (*word translation model*), $n()$ (*fertility model*), $d_1()$ (*head distortion model*) and $d_{>1}()$ (*non-head distortion model*) and the language model employs the distribution *tri*() (*trigram model*).

For IBM-4 and trigram language model, we have:

$$\mathcal{T}_i = \prod_{k=1}^{\phi_i} t(\tau_{ik}|e_i)$$

$$\mathcal{N}_i = \begin{cases} n_0\left(\phi_0 | \sum_{i=1}^{l} \phi_i\right) & \text{if } i = 0 \\ \phi_i!\; n(\phi_i|e_i) & \text{if } 1 \le i \le l \end{cases}$$

$$\mathcal{D}_i = \begin{cases} 1 & \text{if } i = 0 \\ \prod_{k=1}^{\phi_i} p_{ik}(\pi_{ik}) & \text{if } 1 \le i \le l \end{cases}$$

$$\mathcal{L}_i = \begin{cases} 1 & \text{if } i = 0 \\ \texttt{tri}(e_i|e_{i-2}e_{i-1}) & \text{if } 1 \le i \le l \end{cases}$$

where,

$$n_0(\phi_0|m') = \binom{m'}{\phi_0} p_0^{m'-\phi_0} p_1^{\phi_0}$$

$$p_{ik} = \begin{cases} d_1(j - c_{\rho_i}|\mathcal{A}(e_{\rho_i}), \mathcal{B}(\tau_{i1})) & \text{if } k = 1 \\ d_{>1}(j - \pi_{ik-1}|\mathcal{B}(\tau_{ik})) & \text{if } k > 1 \end{cases}$$

$$\rho_i = \max_{i'<i} \{i' : \phi_{i'} > 0\}$$

$$c_\rho = \left\lceil \frac{1}{\phi_\rho} \sum_{k=1}^{\phi_\rho} \pi_{\rho k} \right\rceil.$$

$\mathcal{A}$ and $\mathcal{B}$ are *word classes*, ρ_i is the previous fertile English word, c_ρ is the center of the French words connected to the English word e_ρ, p_1 is the probability of connecting a French word to the null word (e_0), and $p_0 = 1 - p_1$.

Although IBM-4 is a complex model, what is important for us in designing an efficient decoding algorithm is the fact that it can be factorized into $\mathcal{T}, \mathcal{D}, \mathcal{N}$and $\mathcal{L}$.

2.3 Alternating Optimization Framework The goal of the decoder is to solve the following search problem:

$$(\hat{\mathbf{e}}, \hat{\mathbf{a}}) = \underset{\mathbf{e},\mathbf{a}}{argmax}\ \ Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right) Pr\left(\mathbf{e}\right)$$

when $Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right)$ and $Pr\left(\mathbf{e}\right)$ have been defined as described in the previous section.

In the alternating optimization framework, instead of joint optimization, we alternate between optimizing $\mathbf{e}$ and $\mathbf{a}$:

$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{argmax}\ \ Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right) Pr\left(\mathbf{e}\right) \tag{2.3}$$

$$\hat{\mathbf{a}} = \underset{\mathbf{a}}{argmax}\ \ Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right) Pr\left(\mathbf{e}\right) \tag{2.4}$$

In the search problem specified by Equation 2.3, we keep the length of the translation (l) and the alignment ($\mathbf{a}$) fixed while in the search problem specified by Equation 2.4, we keep the translation ($\mathbf{e}$) fixed. We start with an initial alignment, and find the best translation for $\mathbf{f}$ with that alignment. Next, keeping the translation fixed we find a new alignment which is at least as good as the previous one. In this manner, we refine both the alignment and the translation iteratively. The framework does not require that the two problems be solved exactly. Suboptimal solutions to the two problems in every iteration are sufficient for the algorithm to make progress.

Alternating optimization framework is useful in designing fast decoding algorithms for the following reason:

LEMMA 2.1. **Fixed Alignment Decoding** *[14]: The solution to the search problem specified by Equation 2.3 can be found in $O(m)$ time by Dynamic Programming.*

In addition, reasonably good suboptimal solution to the search problem specified by Equation 2.4 can be computed in $O\left(m\right)$ by local search [14].

3 Searching a Family of Alignments

In this section, we describe how to construct a family of alignments starting with any alignment.

3.1 Alignment Transformation Operations Let $\mathbf{a}, \mathbf{a}'$ be any two alignments. Let (τ, π) and $(\tau',\ \pi')$ be the tableau and permutation induced by $\mathbf{a}$ and $\mathbf{a}'$ respectively. We define a relation R between alignments and say that $\mathbf{a}'R\ \mathbf{a}$ if $\mathbf{a}'$ can be derived from $\mathbf{a}$ by performing one of the operations `COPY`, `GROW`, `SHRINK` and `MERGE` on each of $(\tau_i, \pi_i),\ 0 \leq i \leq l$ starting with (τ_1, π_1). Let i and i' be the counters for (τ, π) and (τ', π') respectively. Initially, $(\tau'_0, \pi'_0) = (\tau_0, \pi_0)$ and $i' = i = 1$. The operations are as follows:

1. `COPY`:
   $(\tau'_{i'}, \pi'_{i'}) \leftarrow (\tau_i, \pi_i)$;
   $i \leftarrow i + 1;\ i' \leftarrow i' + 1$.

2. `GROW`:
   $(\tau'_{i'}, \pi'_{i'}) \leftarrow (\{\}, \{\})$;
   $(\tau'_{i'+1}, \pi'_{i'+1}) \leftarrow (\tau_i, \pi_i)$;
   $i \leftarrow i + 1;\ i' \leftarrow i' + 2$.

3. `SHRINK`:
   $(\tau'_0, \pi'_0) \leftarrow (\tau'_0 \cup \tau_i, \pi'_0 \cup \pi_i)$;
   $i \leftarrow i + 1$.

4. `MERGE`:
   $(\tau'_{i'-1}, \pi'_{i'-1}) \leftarrow (\tau'_{i'-1} \cup \tau_i, \pi'_{i'-1} \cup \pi_i)$;
   $i \leftarrow i + 1$.

Figure 3 illustrates the alignment transformation operations on an alignment and the resulting alignment.

The four alignment transformation operations allow us to generate alignments that are related to the starting alignment but have some structural difference. The `COPY` operations maintain structural similarity in some parts between the starting alignment and the new alignment. The `GROW` operations increase the size of the alignment and therefore, the length of the translation. The `SHRINK` operations reduce the size of the alignment and therefore, the length of the translation. `MERGE` operations increase the fertility of words.

3.2 A Family of Alignments Given an alignment $\mathbf{a}$, the relation R defines the following family of alignments: $\mathbf{A} = \{\mathbf{a}' :\ \mathbf{a}'R\ \mathbf{a}\}$. Further, if $\mathbf{a}$ is one-to-one, the size of this family of alignments is $|\mathbf{A}| = \Theta\left(4^m\right)$. We call $\mathbf{a}$ the *generator* of the family $\mathbf{A}$.

The key idea in our decoding algorithm is to find a good family of alignments $\mathbf{A}$ and to compute the optimal solution in this family:

$$(\hat{\mathbf{e}}, \hat{\mathbf{a}}) = \underset{\mathbf{e},\ \mathbf{a}\in\mathbf{A}}{argmax}\ \ Pr\left(\mathbf{f}, \mathbf{a}|\mathbf{e}\right) Pr\left(\mathbf{e}\right) \tag{3.5}$$

3.3 A Dynamic Programming Algorithm We now discuss how to compute the optimal solution in a family of alignments.

LEMMA 3.1. *The solution to the search problem specified by Equation 3.5 can be computed in $O\left(m^2\right)$ time by Dynamic Programming when $\mathbf{A}$ is a family of alignments as defined in Section 3.2.*

The dynamic programming algorithm builds a set of hypotheses and reports the hypothesis with the best score and the corresponding translation, tableau and permutation. The algorithm works in m phases and in each phase it constructs a set of partial hypotheses

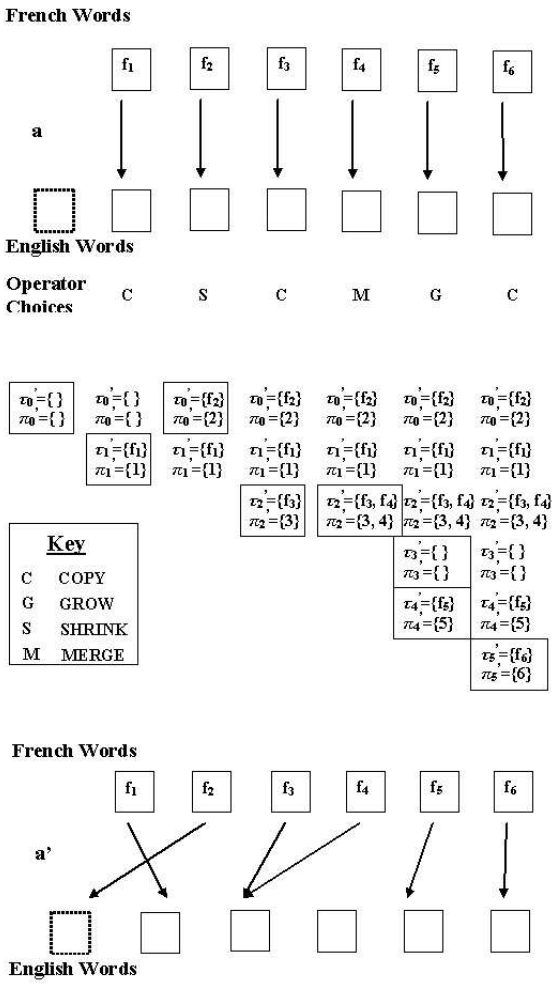

Figure 3: Alignment Transformation Operations

by expanding the partial hypotheses from the previous phase. A *partial hypothesis* after the ith phase, h, is a tuple $(e_0 \dots e_{i'}, \tau'_0 \dots \tau'_{i'}, \pi'_0 \dots \pi'_{i'}, C)$ where $e_0 \dots e_{i'}$ is the partial translation, $\tau'_0 \dots \tau'_{i'}$ is the partial tableau, $\pi'_0 \dots \pi'_{i'}$ is the partial permutation, and C is the score of the partial hypothesis. In the beginning of the first phase, there is only one partial hypothesis $(e_0, \tau'_0, \pi'_0, 0)$. In the ith phase, we extend a hypothesis as follows:

1. Do an alignment transformation operation on the pair (τ_i, π_i).

2. For each pair $(\tau'_{i'}, \pi'_{i'})$ added by doing the operation

   (a) Choose a word $e_{i'}$ from the English vocabulary.

   (b) Include $e_{i'}$ and $(\tau'_{i'}, \pi'_{i'})$ in the partial hypothesis.

3. Update the score of the hypothesis.

As observed in Section 3.2, an alignment transformation operation can result in the addition of 0 or 1 or 2 new tablets. Since each tablet corresponds to an English word, the expansion of a partial hypothesis results in appending 0 or 1 or 2 new words to the partial sentence:

1. `COPY`: An English word $e_{i'}$ is appended to the partial translation (i.e. the partial translation grows from $e_0 \dots e_{i'-1}$ to $e_0 \dots e_{i'}$). The word $e_{i'}$ is chosen from the set of candidate translations of the French words in the tablet τ_i. If we assume that the number of candidate translations a French word can have in the English vocabulary is bounded by N_F, then the number of new partial hypotheses resulting from the `COPY` operation is at most N_F.

2. `GROW`: Two English words $e_{i'}, e_{i'+1}$ are appended to the partial translation as a result of which the partial translation grows from $e_0 \dots e_{i'-1}$ to $e_0 \dots e_{i'} e_{i'+1}$. The word $e_{i'}$ is chosen from the set of infertile English words and $e_{i'+1}$ from the set of English translations of the French words in the tablet τ_i. If the number of infertile words in the English vocabulary is N_0, then the number of new partial hypotheses resulting from the `GROW` operation is at most $N_F N_0$.

3. `SHRINK, MERGE`: The partial translation remains unchanged. Only one new partial hypothesis is generated.

Figure 4 illustrates the expansion of a partial hypothesis using the alignment transformation operations.

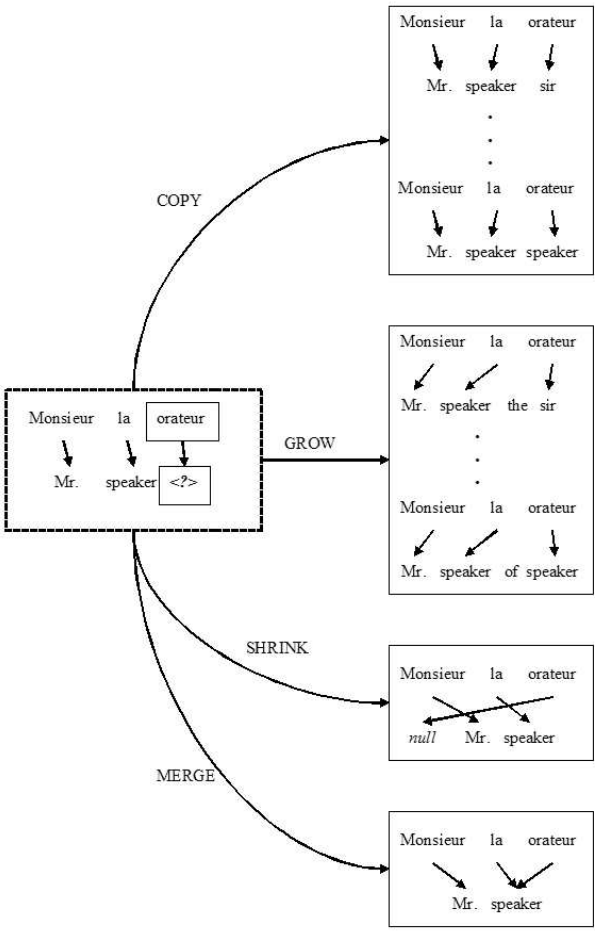

Figure 4: Partial Hypothesis Expansion

At the end of a phase of expansion, we have a set of partial hypotheses. These hypotheses can be put into classes based on the following:

1. The last two words in the partial translation $(e_{i'-1}, e_{i'})$,
2. Fertility of the last word in the partial translation $(|\pi'_{i'}|)$ and
3. The center of the tablet corresponding to the last word in the partial translation.

If two partial hypotheses in the same class are extended using the same operation, then their scores increase by equal amount. Therefore, for each class of hypotheses the algorithm retains only the one with the highest score.

3.3.1 Analysis The algorithm has m phases and in each phase a set of partial hypotheses are expanded. The number of partial hypotheses generated in any phase is bounded by the product of the number of hypothesis classes in that phase and the number of partial hypotheses yielded by the alignment transformation operations. We first count the number of partial hypotheses classes in phase i. There are at most $|V_E|^2$ choices for $(e_{i'-1}, e_{i'})$, at most ϕ_{max} choices for the fertility of $e_{i'}$ and m choices for the center of the tablet corresponding to $e_{i'}$. Therefore, the number of partial hypotheses classes in phase i is at most $\phi_{max}\,|V_E|^2\,m$. The alignment transformation operations on a partial hypothesis result in atmost $N_F(1+N_0)+2$ new partial hypotheses. Therefore, the number of partial hypotheses generated in phase i is at most $\phi_{max}(N_F(1+N_0)+2)\,|V_E|^2\,m$. As there are totally m phases, the total number of partial hypotheses generated by the algorithm is at most $\phi_{max}(N_F(1+N_0)+2)\,|V_E|^2\,m^2$. It must be noted that ϕ_{max}, N_F and N_0 are constants independent of the length of the French sentence. Therefore, the number of operations in the algorithm is $O\left(m^2\right)$. In practice $\phi_{max} < 10$, $N_F \leq 10$, and $N_0 \leq 100$.

3.4 Iterative Search Algorithm We can explore several alignment families iteratively using the alternating optimization framework. In each iteration we solve two problems. In the first problem, we start with a generator alignment **a**, build an alignment family **A** for the generator, and find the best solution in that family using the dynamic programming algorithm. In the second problem, we find a new generator for the next iteration. In order to find a new generator, we swap the tablets in the solution found in the previous step and check if that improves the score. In fact, we find the best swap of tablets that improves the score of the

solution. Clearly, the resulting alignment $\tilde{\mathbf{a}}$ is not part of the alignment family $\mathbf{A}$. We use $\tilde{\mathbf{a}}$ as the generator in the next iteration.

3.5 Pruning Although our dynamic programming algorithm takes $O\left(m^2\right)$ time to compute the translation, the constant in the O is prohibitively large. In practice, the number of partial hypotheses generated by the algorithm is substantially smaller than the bound in Section 3.3.1, but large enough to make the algorithm slow. In this section we describe two partial hypothesis pruning schemes which are helpful in speeding up the algorithm.

3.5.1 Pruning with the Geometric Mean At each phase of the algorithm, the geometric mean of the scores of partial hypotheses generated in that phase is computed. Only those partial hypotheses whose scores are at least as good as the geometric mean are retained for the next phase and the rest are discarded. Although conceptually very simple, pruning the partial hypotheses with the Geometric Mean as the cutoff is a very efficient pruning scheme as demonstrated by our empirical results.

3.5.2 Generator Guided Pruning In this scheme, we take the generator of the alignment family $\mathbf{A}$ and find the best translation (and tableau and permutation) using the $O\left(m\right)$ algorithm for Fixed Alignment Decoding. We then determine the score $C^{(i)}$, at each of the m phases, of the hypothesis that generated the optimal solution. These scores are used to prune the partial hypotheses of the dynamic programming algorithm. In the ith phase of the algorithm, only those partial hypotheses whose scores are at least $C^{(i)}$ are retained for the next phase and the rest are discarded. This pruning strategy incurs the overhead of running the algorithm for Fixed Alignment Decoding for the computation of the cutoff scores. However, this overhead is insignificant in practice.

3.6 Caching The probability distributions ($n, d_1, d_>, t$ and tri) are loaded into memory by the algorithm before decoding. The data is large and is organized in the C++ *hash* and *map* data structures (STL). However, it is better to cache the most frequently used data in smaller data structures so that subsequent accesses are relatively faster. It was observed by profiling the code that caching $d_1, d_>$ and tri is more beneficial than caching n and t.

3.6.1 Caching of Language Model While decoding the French sentence, we know a priori the set of all trigrams that could potentially be accessed by the algorithm. This is because these trigrams are formed by the set of all candidate English translations of the French words in the sentence and the set of infertile words. Therefore, we can assign a unique id for every such trigram. When the trigram is accessed for the first time, we store it in an array indexed by its id. Subsequent accesses to the trigram make use of the cached value.

3.6.2 Caching of Distortion Model As with the language model, the actual number of distortion probability data values accessed by the decoder while translating a sentence is relatively small compared to the total number of distortion probability data values. Further, distortion probabilities are not dependent on the French words but on the position of the words in the French sentence. Therefore, while translating a batch of sentences of roughly the same length, the same set of data is accessed repeatedly. We cache the distortion probabilities required by the algorithm in a *map* data structure.

4 Experiments

4.1 Experimental Setup In this section, we present the results from our experiments. We designed our experiments to study the following:

1. Effectiveness of the pruning techniques.
2. Effect of caching on the performance.
3. Effectiveness of the alignment transformation operations.
4. Effectiveness of the iterative search scheme.

We used Fixed Alignment Decoding as the baseline algorithm in our experiments. To compare the performance of our algorithm with a state-of-the-art decoding algorithm, we used the implementation of the Greedy decoder [15]. In the empirical results from our experiments, in place of the translation score, we report the logscore (i.e. negative logarithm) of the translation score. When reporting scores for a set of sentences, we treat the geometric mean of their translation scores as the statistic of importance and report the average logscore.

4.1.1 Training of the Models We built a French-English translation model (IBM-4) by training over a corpus of 100 K sentence pairs from the Hansard corpus. The translation model was built using the GIZA++ toolkit [16], [19]. There were 80 word classes which were determined using the *mkcls* tool [17]. We built an English trigram language model by training over a

corpus of 100 K English sentences. We used the CMU-Cambridge Statistical Language Modeling Tool Kit v2 for training the language model [18]. While training the translation and language models, we used the default setting of the corresponding tools. The corpora used for training the models were tokenized using an in-house Tokenizer.

4.1.2 Test Data The data used in the experiments consisted of 11 sets of 100 French sentences picked randomly from the French part of the Hansard corpus. The sets were formed based on the number of words in the sentences. We chose 11 sets of sentences whose length was in the range $6-10, 11-15, \ldots, 56-60$.

4.2 Decoder Implementation We implemented our algorithm in C++ and compiled it using gcc with -O3 optimization setting. Methods which had less than 15 lines of code were inlined.

4.2.1 System The experiments were conducted on an Intel Dual Processor machine (2.6 GHz CPU, 2GB RAM) with Linux as the OS. No other job was running on the machine when the decoder was running.

4.3 Starting Generator Alignment Our algorithm requires a starting alignment to serve as the generator for the family of alignments. We used the alignment $a_j = j$, i.e., $l = m$ and $\mathbf{a} = (1, \ldots, m)$ as the starting alignment. This particular alignment is a natural choice for French and English as their word orders are closely related.

4.4 Effect of Pruning The following measures are indicative of the effectiveness of pruning:

1. Percentage of partial hypotheses retained by the pruning technique at each phase of the dynamic programming algorithm.
2. Time taken by the algorithm for decoding.
3. Logscores of the translations.

4.4.1 Pruning with the Geometric Mean (PGM) Figure 5 shows the percentage of partial hypotheses retained at each phase of the dynamic programming algorithm for a set of 100 French sentences of length 25 when the geometric mean of the scores was used for pruning. With this pruning technique, the algorithm removes more than half (about 55% of the partial hypotheses at each phase).

4.4.2 Generator Guided Pruning (GGP) Figure 6 shows the percentage of partial hypotheses retained at each phase of the dynamic programming algorithm for a set of 100 French sentences of length 25 by the Generator Guided Pruning technique. This pruning technique is very conservative and retains only a small fraction of the partial hypotheses at each phase. All the partial hypotheses that survive in a phase are guaranteed to have scores at least as good as the score of the partial hypothesis corresponding to the Fixed Alignment Decoding solution. On an average, only 5% of the partial hypotheses move to the next phase.

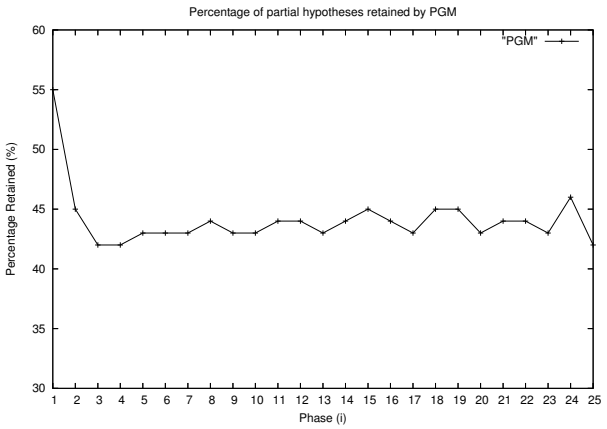

Figure 5: Percentage of hypotheses retained by pruning with geometric mean

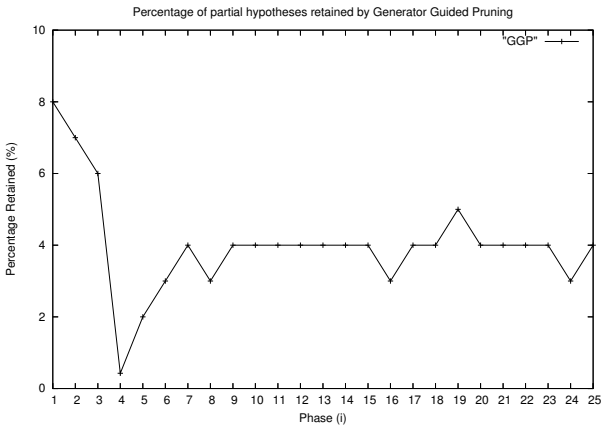

Figure 6: Percentage of partial hypotheses retained by the Generator Guided Pruning technique

4.4.3 Performance Figure 7 shows the time taken by the dynamic programming algorithm with each of the pruning techniques. As hinted by the statistics shown in Figures 6 and 5, the Generator Guided Pruning technique speeds up the algorithm much more than pruning with the geoemtric mean.

Figure 8 shows the logscores of the translations

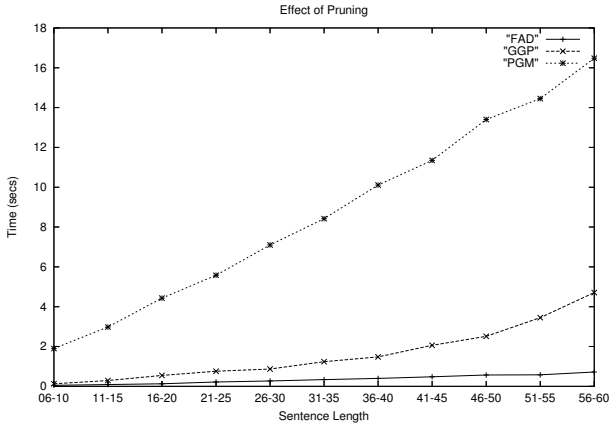

Figure 7: Pruning with Geometric Mean (PGM), Generator Guided Pruning (GGP) and Fixed Alignment Decoding (FAD): Time

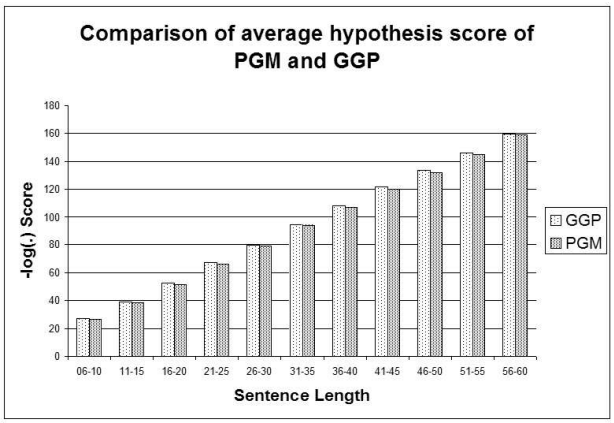

Figure 8: PGM and GGP: Logscores

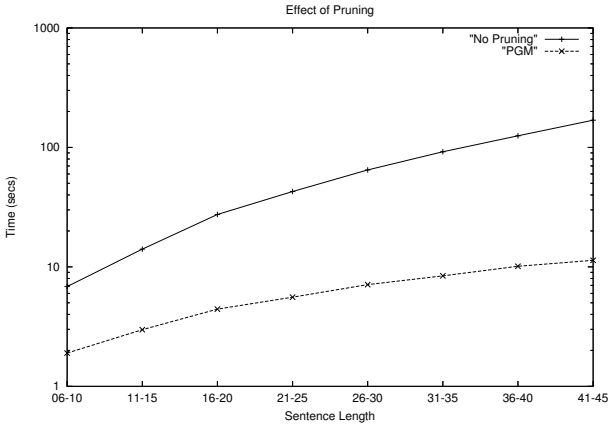

Figure 9: PGM vs No pruning: Time

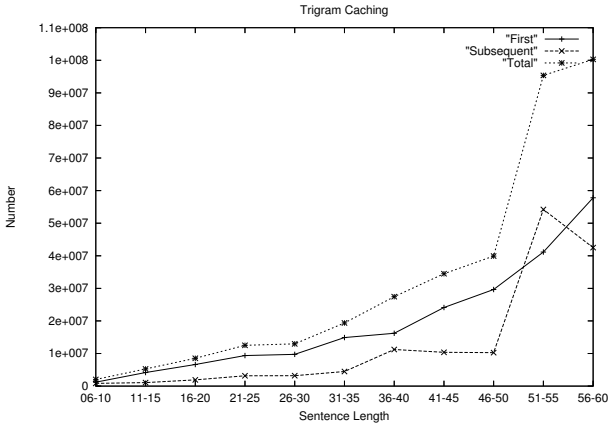

Figure 10: Trigram accesses: First hits, subsequent hits and total hits

found by the algorithm with each of the pruning techniques. Pruning with the Geometric Mean fares better than Generator Guided Pruning, but the difference is not significant.

We also compared the logscores of the translations found by PGM with those of the translations found by the dynamic programming algorithm without pruning and found that the logscores were identical. This means that our pruning techniques are very effective in identifying and removing inconsequential partial hypotheses. Figure 9 shows the time taken by the decoding algorithm when there is no pruning.

From Figure 7 and 8, we conclude that Generator Guided Pruning is a very effective pruning technique.

4.5 Effect of Caching In caching, the number of cache hits is a measure of the repeated use of the cached data. Also of interest is the improvement in runtime due to caching.

4.5.1 Language Model Caching Figure 10 shows the number of distinct trigrams accessed by the algorithm and the number of subsequent accesses to the cached values of these trigrams. We observe that on an average every second trigram is accessed at least once more. Figure 11 shows the time taken for decoding when only the language model is cached. Caching of language model has little effect on smaller length sentences. But as the sentence length grows, caching of language model improves the speed.

4.5.2 Distortion Model Caching Figure 12 shows the counts of first hits and subsequent hits for distortion model values accessed by the algorithm. 99.97% of the total number of accesses are to the cached values. Thus, cached distortion model values are used repeatedly by the algorithm. Figure 11 shows the time taken for

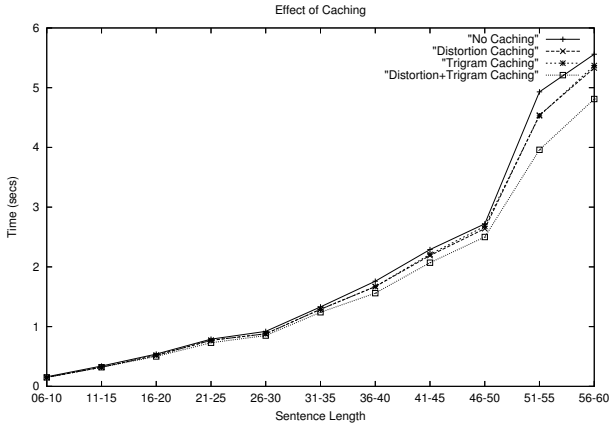

Figure 11: Time taken by GGP with a. No Caching (No caching), b. Distortion Caching (Dist caching), c. Trigram Caching (Trigram caching) and d. Distortion and Trigram Caching (Dist+Trigram caching)

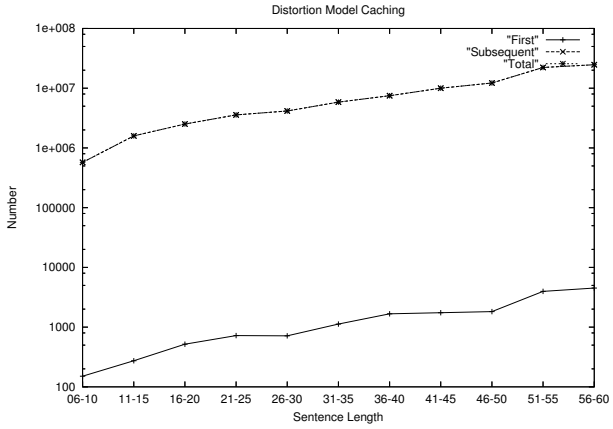

Figure 12: Distortion probability access log

decoding when only the distortion model is cached. Improvement in speed is more significant for longer sentences than for shorter sentences as expected.

Figure 11 shows the time taken for decoding when both the models are cached. As can be observed from the plots, caching of both the models is more beneficial than caching them individually. Although the improvement in speed due to caching is not substantial in our implementation, our experiments do show that cached values are accessed subsequently. We believe that it should be possible to speed up the algorithm further by using better data structures for the cached data.

4.6 Alignment Transformation Operations To understand the effect of the alignment transformation operations on the performance of the algorithm we

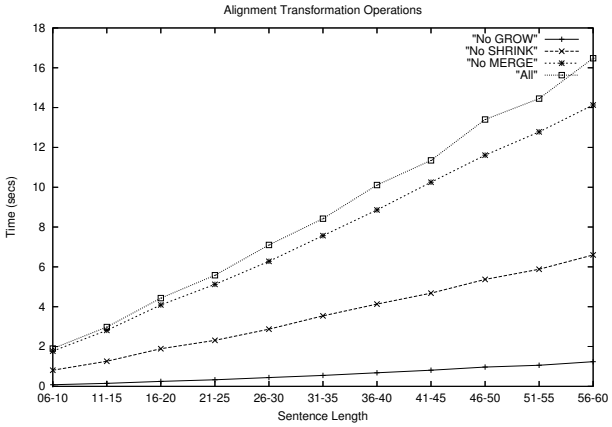

Figure 13: Time a. using all the operators (All Ops), b. not using `GROW`, c. not using `SHRINK` and d. not using `MERGE`

conducted experiments in which we removed each of `GROW`, `MERGE` and `SHRINK` operations and ran the decoder with Generator Guided Pruning.

Figure 14 shows the logscores when the decoder worked with only (`GROW`, `MERGE`, `COPY`) operations, (`SHRINK`, `MERGE`, `COPY`) operations and (`GROW`, `SHRINK`, `COPY`) operations. The logscores are compared with those of the decoder which worked with all the four operations. We observe that the logscores are affected very little by the absence of `SHRINK` operation. However, the absence of `MERGE` operation results in poorer scores. The absence of `GROW` operation also results in poorer scores but the loss is not as significant as with `MERGE`.

Figure 13 shows the time taken for decoding in this experiment. We see that the absence of `MERGE` does not affect the time taken for decoding significantly. We also see that the absence of either `GROW` or `SHRINK` has significant affect on the time taken for decoding. This is not very surprising because `GROW` operations add the highest number of partial hypotheses at each phase of the algorithm 3.3.1. Although a `SHRINK` operation adds only one new partial hypothesis, its contribution to the number of distinct hypothesis classes is significant.

We note that `MERGE` operation while not contributing significantly to the runtime of the algorithm plays an important role in improving the scores.

4.7 Iterative Search Figures 15 and 17 show the time taken by the iterative search algorithm with Generator Guided Pruning (IGGP) and pruning with Geometric Mean (IPGM). Figures 16 and 18 show the corresponding logscores. We observe that the improvement in logscores due to iterative search is not significant. The results highlight the need for a more effective tech-

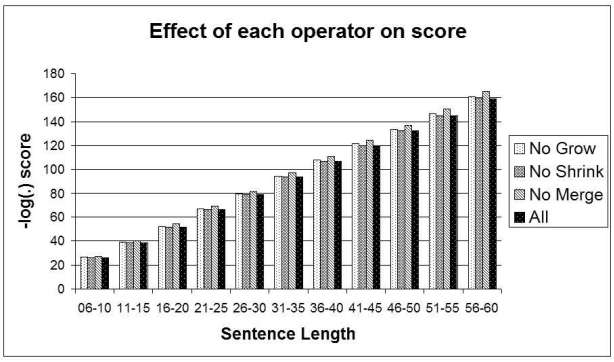

Figure 14: Logscores a. using all the operators (All Ops), b. not using `GROW` (No Grow), c. not using `SHRINK` (No Shrink) and d. not using `MERGE` (No Merge)

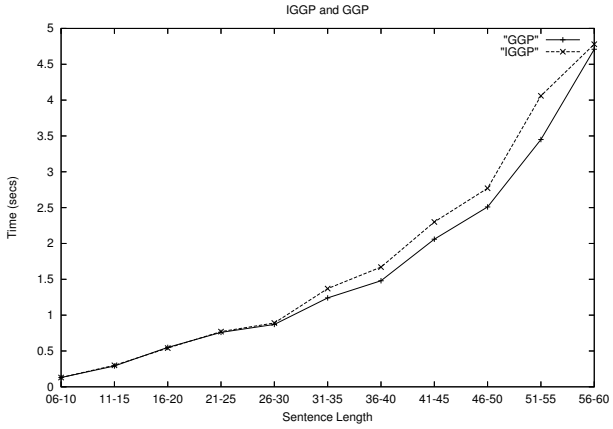

Figure 15: IGGP vs GGP: Time

nique for finding subsequent generators in the iterative algorithm.

4.8 Comparison with the Greedy Decoder Finally, we compare the performance of our algorithm with that of the Greedy decoder. Figure 19 compares the time taken for decoding by our algorithm and the Greedy decoder while Figure 20 shows the logscores. We see that the iterated search algorithm that prunes with the Geometric Mean (IPGM) is faster than the Greedy decoder for sentences whose length is greater than 25. However, the iterated search algorithm that uses Generator Guided Pruning technique (IGGP) is faster than the Greedy decoder for sentences whose length is greater than 10. As can be noted from the plots, IGGP is at least 10 times faster than the greedy algorithm for most sentence lengths.

Our logscores are better than those of the greedy decoder with either of the pruning techniques (Figure 20).

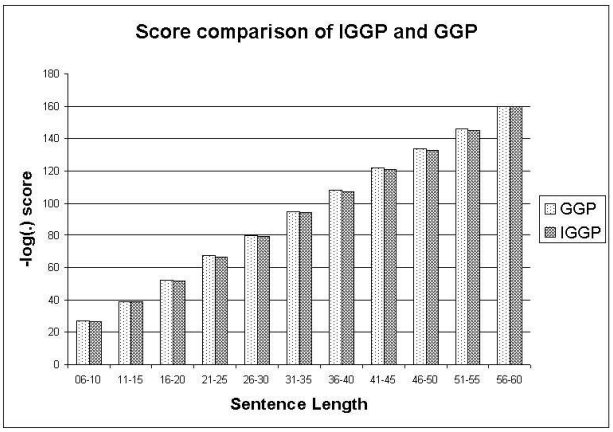

Figure 16: IGGP vs GGP: Logscores

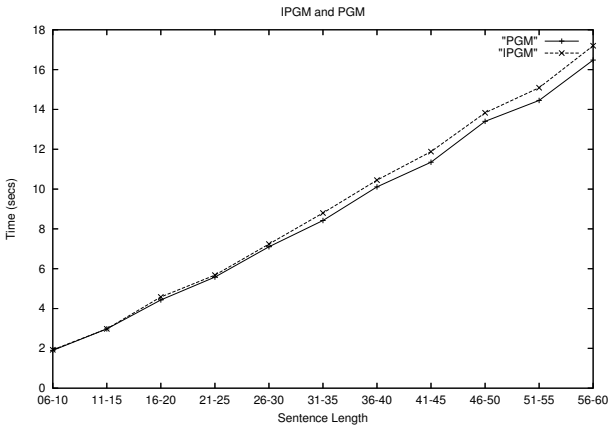

Figure 17: IPGM vs PGM: Time

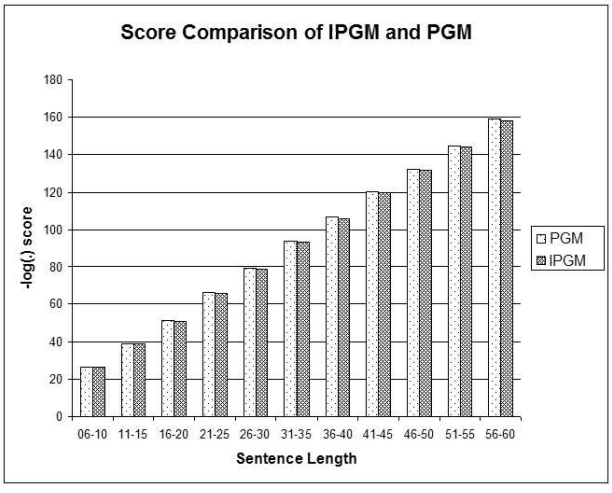

Figure 18: IPGM vs PGM: Logscores

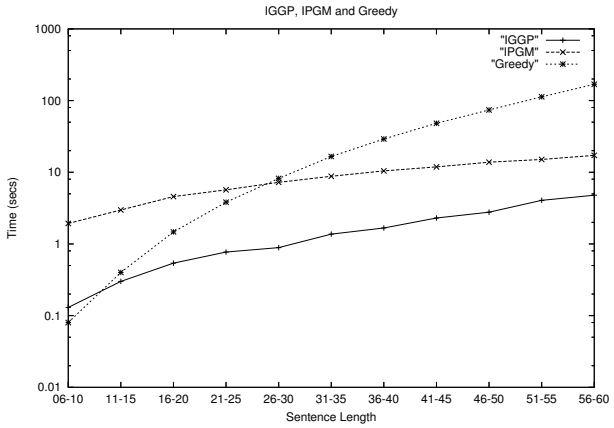

Figure 19: Comparison with the Greedy Decoder: Time

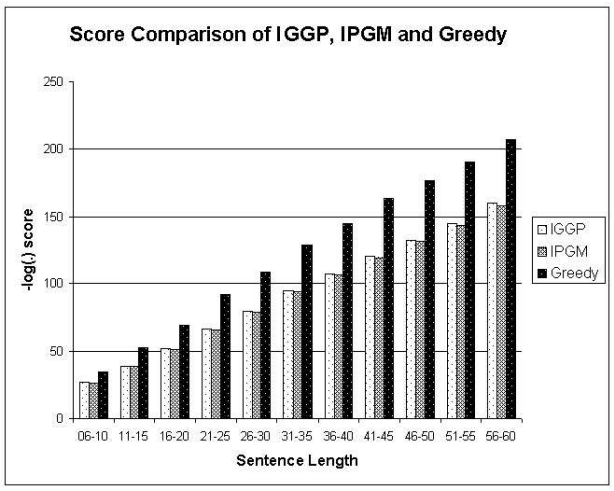

Figure 20: Comparison with the Greedy Decoder: Logscores

5 Conclusions

Searching for a good translation in a family of alignments seems promising as the empirical results indicate. It would be interesting to explore whether good generators can be determined easily from the starting generator. The result of the search in the current alignment family may give good hints for choosing the next generator. Efficient caching of model values is another area that needs to be explored.

6 Acknowledgements

We are grateful to Raghuram Krishnapuram for his help during the course of this work. Special thanks to Prateek Jain for helpful comments on a draft of this paper.

References

[1] K. Knight, *Decoding Complexity in Word-Replacement Translation Models*, Computational Linguistics, 25(4) 1999.

[2] U. Germann, M. Jahr, K. Knight, D. Marcu, and K. Yamada, *Fast Decoding and Optimal Decoding for Machine Translation*, Artificial Intelligence, 2003.

[3] P. Brown, S. Della Pietra, V. Della Pietra, and R. Mercer, *The Mathematics of Machine Translation: Parameter Estimation*, Computational Linguistics, 16(2), pp. 263-311, 1993.

[4] C. Tillman, *Word Re-ordering and Dynamic Programming based Search Algorithm for Statistical Machine Translation*, Ph.D. Thesis, University of Technology Aachen, 2001, pp. 42-45.

[5] M. Held and R. Karp, *A Dynamic Programming Approach to Sequencing Problems*, J. SIAM, 10(1), pp. 196-210, 1962.

[6] U. Germann, *Greedy Decoding for Statistical Machine Translation in Almost Linear Time*, in Proceedings of HLT-NAACL, 2003.

[7] F. Och, N. Ueffing, and H. Ney, *An efficient A* search algorithm for statistical machine translation*, in Proceedings of the ACL 2001 Workshop on Data-Driven Methods in Machine Translation, pp. 55-62.

[8] C. Tillman and H. Ney, *Word reordering and DP-based search in statistical machine translation*, in Proceedings of the 18th COLING, pp. 850-856, 2000.

[9] Y. Wang and A. Waibel, *Decoding algorithm in statistical machine translation*, in Proceedings of the 35th ACL, pp. 366-372, 1997.

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

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

[12] Y. Al-Onaizan, J. Curin, M. Jahr, K. Knight, J. Lafferty, D. Melamed, F. Och, D. Purdy, N. Smith, and D. Yarowsky, *Statistical Machine Translation: Final Report*, JHU Workshop, 1999.

[13] R. Udupa and T. Faruquie, *An English-Hindi Statistical Machine Translation System*, in Proceedings of the 1st IJCNLP, pp. 626-632, 2004.

[14] R. Udupa, T. Faruquie, and H. Maji, *An Algorithmic Framework for the Decoding Problem in Statistical Machine Translation*, in Proceedings of the 20th COLING, 2004.

[15] D. Marcu and F. Och, Greedy Decoder for Statistical Machine Translation, `http://www.isi.edu/licensed-sw/rewrite-decoder/`

[16] F. Och, *GIZA++*, `http://www-i6.informatik.rwth-aachen.de/Colleagues/och/software/GIZA++.html`

[17] F. Och, *mkcls*, `http://www-i6.informatik.rwth-aachen.de/Colleagues/och/software/mkcls.html`

[18] R. Rosenfeld and P. Clarkson, *The CMU-Cambridge Statistical Language Modeling Toolkit v2*, `http://mi.eng.cam.ac.uk/~prc14/toolkit_documentation.html`

[19] F. Och and H. Ney, *Improved Statistical Alignment Models*, in Proceedings of the 38th ACL, Hongkong, China, pp. 440–447, 2000.