A Scalable Decoder for Parsing-based Machine Translation with Equivalent Language Model State Maintenance

Zhifei Li and **Sanjeev Khudanpur**
Department of Computer Science and Center for Language and Speech Processing
Johns Hopkins University, Baltimore, MD 21218, USA
zhifei.work@gmail.com and khudanpur@jhu.edu

Abstract

We describe a scalable decoder for parsing-based machine translation. The decoder is written in JAVA and implements all the essential algorithms described in Chiang (2007): chart-parsing, m-gram language model integration, beam- and cube-pruning, and unique k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable. We also propose an algorithm to maintain equivalent language model states that exploits the *back-off* property of m-gram language models: instead of maintaining a separate state for each distinguished sequence of "state" words, we merge multiple states that can be made *equivalent* for language model probability calculations due to back-off. We demonstrate experimentally that our decoder is more than 30 times faster than a baseline decoder written in PYTHON. We propose to release our decoder as an open-source toolkit.

1 Introduction

Large-scale parsing-based statistical machine translation (MT) has made remarkable progress in the last few years. The systems being developed differ in whether they use source- or target-language syntax. For instance, the hierarchical translation system of Chiang (2007) extracts a synchronous grammar from pairs of strings, Quirk et al. (2005), Liu et al. (2006) and Huang et al. (2006) perform syntactic analyses in the source-language, and Galley et al. (2006) use target-language syntax.

A critical component in parsing-based MT systems is the decoder, which is complex to implement and scale up. Most of the systems described above employ tailor-made, dedicated decoders that are not open-source, which results in a high barrier to entry for other researchers in the field. However, with the algorithms proposed in (Huang and Chiang, 2005; Chiang, 2007; Huang and Chiang, 2007), it is possible to develop a general-purpose decoder that can be used by all the parsing-based systems. In this paper, we describe an important first-step towards an extensible, general-purpose, scalable, and open-source parsing-based MT decoder. Our decoder is written in JAVA and implements all the essential algorithms described in Chiang (2007): chart-parsing, m-gram language model integration, beam- and cube-pruning, and unique k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable.

Straightforward integration of an m-gram language model (LM) into a parsing-based decoder substantially increases its computational complexity. Therefore, it is important to develop efficient methods for LM integration. We propose an algorithm to maintain equivalent LM states by exploiting the *back-off* property of m-gram LMs. Specifically, instead of maintaining a separate state for each distinguished sequence of "state" words, we merge multiple states that can be made *equivalent* for LM calculations by anticipating such back-off.

We demonstrate experimentally that our decoder is 38 times faster than a previous decoder written in PYTHON. Furthermore, the distributed computing permits improving translation quality via large-scale LMs. We have successfully use our decoder to translate about a million sentences in a parallel corpus for large-scale discriminative training experiments.

2 Parsing-based MT Decoder

In this section, we discuss the core algorithms implemented in our decoder. These algorithms have been discussed by Chiang (2007) in detail, and we recapitulate the essential parts here for completeness.

2.1 Grammar Formalism

Our decoder assumes a probabilistic synchronous context-free grammar (SCFG). Following the notation in Venugopal et al. (2007), a probabilistic SCFG comprises a set of source-language terminal symbols T_S, a set of target-language terminal symbols T_T, a shared set of nonterminal symbols N, and a set of rules of the form

$$X \rightarrow \langle \gamma, \alpha, \sim, w \rangle \,, \tag{1}$$

where $X \in N$, $\gamma \in [N \cup T_S]^*$ is a (mixed) sequence of nonterminals and source terminals, $\alpha \in [N \cup T_T]^*$ is a sequence of nonterminals and target terminals, $\sim$ is a one-to-one correspondence or *alignment* between the nonterminal elements of γ and α, and $w \geq 0$ is a weight assigned to the rule. An illustrative rule for Chinese-to-English translation is

$$NP \;\rightarrow\; \langle\, NP_0 \text{ 的 } NP_1 \,,\, NP_1 \text{ of } NP_0 \,\rangle \,,$$

where the Chinese word 的 (pronounced *de* or *di*) means *of*, and the alignment, encoded via subscripts on the nonterminals, causes the two noun phrases around 的 to be reordered around *of* in the translation. The rule weight is omitted in this example.

A bilingual SCFG derivation is analogous to a monolingual CFG derivation. It begins with a pair of *aligned* start symbols. At each step, an *aligned* pair of nonterminals is rewritten as the two corresponding components of a single rule. In this sense, the derivations are generated synchronously.

Our decoder presently handles SCFGs of the kind extracted by Heiro (Chiang, 2007), but is easily extensible to more general SCFGs and closely related formalisms such as synchronous tree substitution grammars (Eisner, 2003; Chiang, 2006).

2.2 MT Decoding as Chart Parsing

Given a source-language sentence f^*, the decoder must find the target-language yield $e(D)$ of the best derivation D among all derivations with source-language yield $f(D) = f^*$, i.e.

$$e^* \;=\; e\left(\arg\max_{D\,:\,f(D)=f^*} w(D)\right) , \tag{2}$$

where $w(D)$ is the composite weight of D.

The parser may be treated as a deductive proof system (Shieber et al., 1995). Formally (cf. (Chiang, 2007)), a parser defines a space of weighted *items*, with some items designated as *axioms* and some as *goals*, and a set of *inference rules* of the form

$$\frac{I_1 : w_1 \;\; \cdots \;\; I_k : w_k}{I : w} \;\; \phi \,,$$

which states that if all the *antecedent* items I_i are provable, respectively with weight w_i, then the *consequent* item I is provable with weight w, provided the side condition ϕ holds. For a grammar with a maximum of two (pairs of) nonterminals per rule[1], Figure 1 illustrates the resulting chart parsing procedure, including the integration of an m-gram LM.

The actual decoding algorithm maintains a *chart*, which contains an array of *cells*. Each cell in turn maintains a list of proved *items*. The parsing process starts with the axioms, and proceeds by applying the inference rules to prove more and more items until a goal item is proved. Whenever the parser proves a new item, it adds the item to the appropriate chart cell. It also maintains backpointers to antecedent items, which are used for k-best extraction, as discussed in Section 2.4 below.

In a SCFG-based decoder, an *item* is identified by its source-language span, left-side nonterminal label, and left- and right-context for the target-language m-gram LM. Therefore, in a given *cell*, the maximum possible number of items is $O(|N||T_T|^{2(m-1)})$, and the worst case decoding complexity is

$$O\left(|N|^K \, |T_T|^{2K(m-1)} n^3\right) , \tag{3}$$

where K is the maximum number of nonterminal pairs per rule and n is the source-language sentence length (Venugopal et al., 2007).

[1]For more general grammars with $K \geq 2$ pairs of nonterminals per rule, see Venugopal et al. (2007).

$$\frac{}{X\to\langle\gamma,\alpha\rangle : w} \quad (X\to\langle\gamma,\alpha,w\rangle)\in G$$

$$\frac{X\to\langle f_{i+1}^{j},\,\alpha\rangle : w}{[X,\,i,\,j;\,q(\alpha)] : wp(\alpha)}$$

$$\frac{Z\to\langle f_{i+1}^{i_1} X f_{j_1+1}^{j},\,\alpha\rangle : w \quad [X,i_1,j_1;e_1] : w_1}{[Z,\,i,\,j;\,q(\alpha')] : ww_1p(\alpha')} \quad \alpha' = \alpha[e_1/X]$$

$$\frac{Z\to\langle f_{i+1}^{i_1} X_1 f_{j_1+1}^{i_2} Y_2 f_{j_2+1}^{j},\,\alpha\rangle : w \quad [X,i_1,j_1;e_1] : w_1 \quad [Y,i_2,j_2;e_2] : w_2}{[Z,\,i,\,j;\,q(\alpha')] : ww_1w_2p(\alpha')} \quad \alpha' = \alpha[e_1/X_1, e_2/Y_2]$$

$$\text{Goal item: } [S,0,n;\langle s\rangle^{m-1}\star e\langle/s\rangle]$$

Figure 1: Inference rules from Chiang (2007) for a parser with an m-gram LM. G denotes the translation grammar. $w[x/X]$ denotes substitution of the string x for the symbol X in the string w. The function $p(\cdot)$ provides the LM probability for all *complete* m-grams in a string, while the function $q(\cdot)$ elides symbols whose m-grams have been accounted for by $p(\cdot)$. Details about the functions $p(\cdot)$ and $q(\cdot)$ are provided in Section 4.

2.3 Pruning in a Decoder

Severe pruning is needed in order to make the decoding computationally feasible for SCFGs with large vocabularies T_T and detailed nonterminal sets. In our decoder, we incorporate two pruning techniques described by (Chiang, 2007; Huang and Chiang, 2007). For *beam pruning*, in each cell, we discard all items whose weight is worse, by a relative threshold β, than the weight of the best item in the same cell. If too many items pass the threshold, a cell only retains the top-b items by weight. When combining smaller items to obtain a larger item by applying an inference rule, we use *cube-pruning* to simulate k-best extraction in each destination cell, and discard combinations that lead to an item whose weight is worse than the best item in that cell by a margin of ϵ.

2.4 k-best Extraction Over Hyper-graphs

For each source-language sentence f^*, the output of the chart-parsing algorithm may be treated as a *hyper-graph* representing a set of likely hypotheses D in (2). Briefly, a hyper-graph is a set of *vertices* and *hyper-edge*s, with each hyper-edge connecting a *set* of antecedent vertices to a consequent vertex, and a special vertex designated as the *target vertex*. In parsing parlance, a vertex corresponds to an item in the chart, a hyper-edge corresponds to a SCFG rule with the nonterminals on the right-side replaced by back-pointers to antecedent items, and the target vertex corresponds to the goal item[2].

Given a hyper-graph for a source-language sentence f^*, we use the k-best extraction algorithm of Huang and Chiang (2005) to extract its k most likely translations. Moreover, since many different derivations D in (2) may lead to the same target-language yield $e(D)$, we adopt the modification described in Huang et al. (2006) to efficiently generate the *unique* k best translations of f^*.

3 Parallel and Distributed Computing

Many applications of parsing-based MT entail the use of SCFGs extracted from millions of bilingual sentence pairs and LMs extracted from billions of words of target-language text. This requires the decoder to make use of *distributed* computing to spread the memory required to load large-scale SCFGs and LMs onto *multiple processors*. Furthermore, techniques such as iterative minimum error-rate training (Och et al., 2003) as well as web-based MT services require the decoder to translate a large number of source-language sentences per unit time. This requires the decoder to make use of *parallel* computing to utilize each *individual multi-core processor* more effectively. We have incorporated two such performance enhancements in our decoder.

[2]In a decoder integrating an m-gram LM, there may be multiple goal items due to different LM contexts. However, one can image a *single* goal item identified by the span $[0, n]$ and the goal nonterminal S, but not by the LM contexts.

3.1 Parallel Decoding

We have enhanced our decoder to translate multiple source-language sentences in parallel by exploiting the ability of a multi-core processor to concurrently run several *threads* that share memory. Specifically, given one (or more) document(s) containing multiple source-language sentences, the decoder automatically splits the set of sentences into several subsets, and initiates concurrent decoding threads; once all the threads finish, the main thread merges back the translations. Since all the threads naturally share memory, the decoder needs to load the (large) SCFG and LM into memory *only once*. This multi-threading provides a very significant speed-up.

3.2 Distributed Language Models

It is not possible in some cases to load a very large LM into memory on a *single* machine, particularly if the SCFG is also very large. In other cases, loading the LM each time the decoder runs may be too time-consuming relative to the time required for decoding itself, such as in iterative decoding with updated combination weights during minimum error-rate training. It is therefore desirable to have dedicated servers to load parts of the LM[3] — an idea that has been exploited by (Zhang et al., 2006; Emami et al., 2007; Brants et al., 2007).

Our implementation can load a (partitioned) LM on different servers before initiating decoding. The decoder remotely calls the servers to obtain individual LM probabilities, and linearly interpolates them on the fly using a given set of interpolation weights. With this architecture, one can deal with a very large target-language text corpus by splitting it into many parts and training separate LMs from each. The run-time interpolation capability may also be used for LM adaptation, e.g. for building document-specific language models.

To mitigate potential network communication delays inherent to a distributed LM, we implement a simple *cache* mechanism in the *decoder*. The cache saves the outcomes of the most recent LM calls, including interpolated LM probabilities; the cache is reset whenever its size exceeds a threshold. We could have maintained a cache at each *LM server* as well; however, the resultant saving is not significant because the *trie* data-structures used to implement m-gram LMs are quite fast relative to the cache lookup overhead.

[3]Similarly, distributing the SCFG is also possible.

4 Equivalent LM-state Maintenance

It is clear from the complexity (3) of the inference rules (Figure 1) that a straightforward integration of an m-gram LM adds a multiplicative factor of $|T_T|^{2K(m-1)}$ to the computational complexity of the decoder, where T_T is the set of target-language terminal symbols. We illustrate in this section how this potentially very large multiplier can be dramatically reduced by exploiting the *structure* of the LM.

4.1 Applying an m-gram LM in the Decoder

Integrating an LM into chart parsing requires two functions $p(\cdot)$ and $q(\cdot)$ (see Figure 1) that operate on strings over $T_T \cup \{\star\}$, where $\star$ is a special "placeholder" symbol for an elided part of a target-language string.

The function $p(e)$ calculates the LM probability of the *complete* m-grams in $e \equiv e_1 \,\ldots\, e_l$, i.e.

$$p(e_1 \,\ldots\, e_l) \;=\; \prod_{m \le i \le l \;\;\&\;\; \star \notin e^{i}_{i-(m-1)}} P_{\text{LM}}(e_i \,|\, h_i)\,, \quad (4)$$

where $h_i = e_{i-(m-1)} \ldots e_{i-1}$ is the $m{-}1$-word "LM history" of the target-language word e_i.

Since the p-probability of e does not include the LM probability for the *partial* m-grams (i.e., the first $(m-1)$ words) of e, the *exact* weights of two items $[X, i, j; e]$ and $[X, i, j; e']$ in the chart are *not* available during the bottom-up pruning of Section 2.3. Therefore, as an approximation, we also compute

$$\hat{p}(e) \;=\; \prod_{k=1}^{\min\{m-1,\,|e|\}} P_{\text{LM}}(e_k \,|\, e_1 \,\ldots\, e_{k-1}), \quad (5)$$

an *estimate* of the LM probability of the $m{-}1$-gram prefix of e. This estimated probability is taken into account for pruning purposes (only).

The function $q(e_1 \,\ldots\, e_l)$ determines the left and right *LM states* that must be maintained for future computation of the *exact* LM probability, respectively, of $e_1 \,\ldots\, e_{m-1}$ and $e_{l+1} \,\ldots\, e_{l+m-1}$.

$$q(e_1 \,\ldots\, e_l) \quad (6)$$

$$= \begin{cases} e_1 \ldots e_l & \text{if } l < m-1, \\ e_1 \ldots e_{m-1} \;\star\; e_{l-(m-2)} \ldots e_l & \text{otherwise.} \end{cases}$$

4.2 Back-off Parameterization of m-gram LMs

While many different methods are popular for estimating m-gram LMs, most store the estimated LM parameters in the ARPA *back-off* file format; using the notation e_i^j to denote a target-language word sequence $e_i\, e_{i+1}\, \ldots\, e_j$, the LM probability calculation is carried out as

$$P_{\text{BO}}(e_m \,|\, e_1^{m-1}) \tag{7}$$
$$= \begin{cases} \pi(e_1^m) & \text{if } e_1^m \in \text{ LM} \\ \beta(e_1^{m-1}) \times P_{\text{BO}}(e_m \,|\, e_2^{m-1}) & \text{otherwise,} \end{cases}$$

where the *lower order* probability $P_{\text{BO}}(e_m \,|\, e_2^{m-1})$ is recursively defined in the same way, and $\beta(e_1^{m-1})$ is the back-off weight of the history. The LM file contains the parameter $\pi(\cdot)$ for each *listed* m-gram, and the parameters $\pi(\cdot)$ and $\beta(\cdot)$ for each listed $\widetilde{m}$-gram, $1 \leq \widetilde{m} < m$; for *unlisted* $\widetilde{m}$-grams, $\beta(\cdot) = 1$ by definition.

Observe from (7) that if e_1^m is *not listed* in the LM, the back-off weight $\beta(\cdot)$ is the same for all words e_m, and the backed-off probability $P_{\text{BO}}(e_m \,|\, \cdot)$ is the same for all words e_1. Furthermore, as m grows, the fraction of possible m-grams actually observed in a training corpus diminishes rapidly.

4.3 The Equivalent LM State of an Item

The maximum possible number of items in a *cell* increases exponentially with the LM order m, as discussed in Section 2.2. With pruning (cf. Section 2.3), we restrict the maximum number of items in each cell to some threshold b. Intuitively, therefore, if we increase the LM order m, we should also increase the beam size b to reduce search errors. This could slow down the decoder significantly.

Recall from the previous subsection, however, that when m increases, the fraction of m-grams that will need to back-off also increases. Moreover, even for modest values of m, the decoder considers many "unseen" m-grams (due to reordering and translation combinations) that do not appear in natural texts, leading to frequent back-off during the LM probability calculation (7). In this subsection, we propose a method to collapse equivalent LM states so that the decoder effectively considers many more items in each cell without increasing beam size.

We merge multiple LM states (6) that already have—or back-off to—the *same* "LM history" in the calculation (7) of LM probabilities, e.g. due to different unlisted m-grams that back-off to the same $m{-}1$-gram. For simplicity, we only consider LM state merging by the function $q(\cdot)$ of (6) when $l \geq m{-}1$.

Though the equivalent LM state maintenance technique is discussed here in the context of a parsing-based MT decoder, it is also applicable to standard left-to-right phrase-based decoders. In particular, the right-side equivalent LM state maintenance proposed in Section 4.3.1 may be used.

4.3.1 Obtaining the Equivalent Right LM State

Recall that the *right* LM state $e_{l-(m-2)}^l$ of e_1^l serves as the "LM history" for calculating the *exact* LM probabilities of the yet-to-be-determined word e_{l+1}. Recall further the computation (7) of $P_{\text{BO}}(e_{l+1} \,|\, e_{l-(m-2)}^l)$.

- If the m-gram $e_{l-(m-2)}^{l+1}$ is *not listed* in the LM for *any* word e_{l+1}, then the LM will back-off to $P_{\text{BO}}(e_{l+1} \,|\, e_{l-(m-3)}^l)$, which does not depend on the word $e_{l-(m-2)}$.
- If the $m{-}1$-gram $e_{l-(m-2)}^l$ also is not listed in the LM, then $\beta(e_{l-(m-2)}^l) = 1$.

If these two conditions hold true, $q(\cdot)$ may safely elide the word $e_{l-(m-2)}$ in (6) no matter what words follow e_1^l. The *right* LM state is thus reduced from $m-1$ words to $m-2$ words.

The argument above can be applied recursively to the resulting right LM state $e_{l-(m-2)+i}^l$, where $i \in [0, m-2]$, leading to the *equivalent right state* computation procedure of Figure 2. The procedure IS-A-PREFIX$(e_1^{\widetilde{m}})$ checks if its argument $e_1^{\widetilde{m}}$ is a prefix of any k-gram listed in the LM, $k \in [\widetilde{m}, m]$.

4.3.2 Obtaining the Equivalent Left LM State

Recall that the *left* LM state e_1^{m-1} of e_1^l is the prefix whose exact LM probability is unknown during bottom-up parsing, and is replaced by the estimated probability $\hat{p}(e_1^{m-1})$ of (5) for pruning purposes. Recall further the computation (7) of $P_{\text{BO}}(e_{m-1} \,|\, e_0^{m-2})$.

- If the m-gram e_0^{m-1} is *not listed* in the LM for *any* word e_0, then it will back-off to

```
EQ-R-STATE $(e^l_{l-(m-2)})$
1 ers $\leftarrow e^l_{l-(m-2)}$
2 for $i \leftarrow 0$ to $m-2$          $\triangleright$ left to right
3     if IS-A-PREFIX $(e^l_{l-(m-2)+i})$
4             break          $\triangleright$ stop reducing ers
5     else
6             ers $\leftarrow e^l_{l-(m-2)+i+1}$     $\triangleright$ reduce state
7 return ers
```

Figure 2: Equivalent Right LM State Computation.

$P_{\text{BO}}(e_{m-1} \mid e_1^{m-2})$, which can be computed right away based on e_1^{m-1} without waiting for the unknown e_0. Moreover, the back-off weight $\beta(e_0^{m-2})$ does not depend on the word e_{m-1}.

Therefore, $q(\cdot)$ may safely elide the word e_{m-1}, and reduce the *left* LM state in (6) from e_1^{m-1} to e_1^{m-2}. Also, $p(\cdot)$ should also co-opt $P_{\text{BO}}(e_{m-1} \mid e_1^{m-2})$ into the *complete* m-gram probability of (4) and $\hat{p}(\cdot)$ should exclude e_{m-1} in (5).

The argument above can again be applied recursively to the resulting left LM state e_1^i, $i \in [1, m-1]$, leading to the *equivalent left state* procedure of Figure 3. The procedure IS-A-SUFFIX$(e_1^{\widetilde{m}})$ checks if $e_1^{\widetilde{m}}$ is a suffix of any listed k-gram in the LM, $k \in [\widetilde{m}, m]$. In Figure 3, fin refers to the probability that can be computed right away based on the state itself, for co-opting into the *complete* m-gram probability of (4) as mentioned above.

4.3.3 Modified Cost Functions for Parsing

When carrying out the reduction of the left and right LM states to their shortest equivalents, the formula (4) for calculating the probability of the *complete* m-grams in an item $[X, i, j; e]$, where $e = e_1^l$, is modified as

$$p(e_1^l) = \text{EQ-L-STATE}(e_1^{m-1}).\mathtt{fin} \times \prod_{m \le i \le l \ \&\ \star \notin e^i_{i-(m-1)}} P_{\text{LM}}(e_i \mid h_i)$$

with the further qualification that some care must be taken later to incorporate the back-off weights of the "LM histories" of the suffix of e_1^{m-1} that went missing due to left LM state reduction.

```
EQ-L-STATE $(e_1^{m-1})$
1 els $\leftarrow e_1^{m-1}$
2 fin $\leftarrow 1$          $\triangleright$ update to final probability $p$
3 for $i \leftarrow m-1$ to $1$          $\triangleright$ right to left
4     if IS-A-SUFFIX$(e_1^i)$
5             break          $\triangleright$ stop reducing els
6     else
7             fin $\leftarrow P_{\text{BO}}(e_i \mid e_1^{i-1}) \times$ fin
8             els $\leftarrow e_1^{i-1}$     $\triangleright$ reduce state
9 return els, fin
```

Figure 3: Equivalent Left LM State Computation.

The estimated probability of the left LM state is modified as

$$\hat{p}(e) = \begin{cases} \hat{p}(e) & \text{if } |e| < m-1 \\ \hat{p}(\text{EQ-L-STATE}(e_1^{m-1}).\mathtt{els}) & \text{otherwise,} \end{cases}$$

with $\hat{p}$ as defined in (5).

Finally, the LM state function is

$$q(e_1^l) = \begin{cases} e_1 \ldots e_l & \text{if } l < m-1 \\ \text{EQ-L-STATE}(e_1^{m-1}).\mathtt{els} \ \star & \\ \quad \text{EQ-R-STATE}(e^l_{l-(m-2)}).\mathtt{ers} & \text{otherwise.} \end{cases}$$

4.3.4 Suffix and Prefix Look-Up

As done in the SRILM toolkit (Stolcke, 2002), a back-off m-gram LM is stored using a reverse *trie* data structure. We store the suffix and prefix information in the same data structure without incurring much additional memory cost. Specifically, the prefix information is stored at the back-off state, while the suffix information is stored as one bit alongside the regular m-gram probability.

5 Experimental Results

In this section, we evaluate the performance of our decoder on a Chinese to English translation task.

5.1 System Training

We use various parallel text corpora distributed by the Linguistic Data Consortium (LDC) for the NIST MT evaluation. The parallel data we select contains about 570K Chinese-English sentence pairs, adding up to about 19M words on each side. To train the English language models, we use the English side of the parallel text and a subset of the English Gigaword corpus, for a total of about 130M words.

We use the GIZA toolkit (Och and Ney, 2000), a suffix-array architecture (Lopez, 2007), the SRILM toolkit (Stolcke, 2002), and minimum error rate training (Och et al., 2003) to obtain word-alignments, a translation model, language models, and the optimal weights for combining these models, respectively.

5.2 Improvements in Decoding Speed

We use a PYTHON implementation of a state-of-the-art decoder as our baseline[4] for decoder comparisons. For a direct comparison, we use exactly the same models and pruning parameters. The SCFG contains about 3M rules, the 5-gram LM explicitly lists about 49M k-grams, $k = 1, 2, \ldots, 5$, and the pruning uses $\beta = 10$, $b = 30$ and $\epsilon = 0.1$.

Decoder	Speed (sec/sent)	BLEU-4	
		MT '03	MT '05
Python	26.5	34.4%	32.7%
Java	1.2	**34.5%**	**32.9%**
Java (parallel)	**0.7**		

Table 1: Decoder Comparison: Translation speed and quality on the 2003 and 2005 NIST MT benchmark tests.

As shown in Table 1, the JAVA decoder (without explicit parallelization) is 22 times *faster* than the PYTHON decoder, while achieving slightly *better* translation quality as measured by BLEU-4 (Papineni et al., 2002). The parallelization further speeds it up by a factor of 1.7, making the parallel JAVA decoder is 38 times faster than the PYTHON decoder.

We have used the decoder to successfully decode about one million sentences for a large-scale discriminative training experiment.

5.3 Impact of a Distributed Language Model

We use the SRILM toolkit to build eight 7-gram language models, and load and call the LMs using a distributed LM architecture[5] as discussed in Section 3.2. As shown in Table 2, the 7-gram distributed language model (DLM) significantly improves translation performance over the 5-gram LM. However, decoding is significantly slower (12.2 sec/sent when using the non-parallel decoder) due to the added network communication overhead.

LM type	# k-grams	MT '03	MT '05
5-gram LM	49 M	34.5%	32.9%
7-gram DLM	310 M	**35.5%**	**33.9%**

Table 2: Distributed language model: the 7-gram LM cannot be loaded alongside the SCFG on a single machine; via distributed computing, it yields significant improvement in BLEU-4 over a 5-gram.

5.4 Utility of Equivalent LM States

To reduce the number of search errors, one may either increase the beam size, or employ techniques such as the equivalent LM state maintenance described in Section 4. In this subsection, we compare the tradeoff between the search effort (measured by decoding time per sentence) and the search quality (measured by the average model cost of the best translation found).

Intuitively, collapsing equivalent LM states is useful only when the language model is very sparse, i.e., most of the evaluated m-grams will need to back-off. A sparse LM is obtained in practice by using a large order m relative to the amount of training data. To test this intuition, we train a 7-gram LM using *only* the English side of the parallel text ($\sim$ 19M words). Figure 4 compares maintenance of the full LM state v/s the equivalent LM state. The beam size b for decoding with equivalent LM states is fixed at 30; it is increased considerably—30, 50, 70, 90, 120, and 150—with the full LM state in an effort to reduce search errors. It is clear from the figure that collapsing items that differ due only to equivalent LM states improves the search quality considerably while actually reducing search effort. This shows the effectiveness of equivalent LM state maintenance.

[4]We are extremely thankful to Philip Resnik at University of Maryland for allowing us the use of their PYTHON decoder as the baseline. Thanks also go to David Chiang who originally implement the decoder.

[5]Since our distributed LM architecture dynamically interpolates multiple LM scores, it cannot yet exploit the equivalent LM state maintenance of Section 4, for different LMs will have different reduced LM states. We will address this in the future.

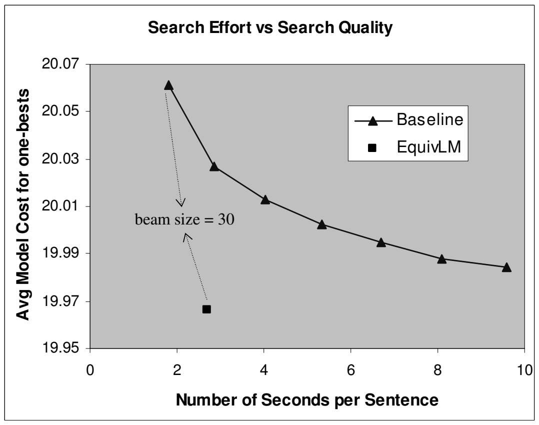

Figure 4: Search quality with equivalent 7-gram LM state maintenance (EquivLM) and without it (Baseline) as a function of search effort as controlled by the beam size.

We also train a 3-gram LM using an English corpus of about 130M words, and repeat the above experiments. In this case, maintaining equivalent LM states costs *more* decoding time than using the full LM state to achieve the same search quality. This is due partly to our inefficient implementation of the prefix- and suffix-lookup required to determine the equivalent LM state, and partly to the fact that with 130M words, a 3-gram LM backs off less frequently.

6 Conclusions

We have described a scalable decoder for parsing-based machine translation. It is written in JAVA and implements all the essential algorithms described in Chiang (2007): chart-parsing, m-gram language model integration, beam- and cube-pruning, and unique k-best extraction. Additionally, parallel and distributed computing techniques are exploited to make it scalable. We demonstrate that our decoder is 38 times faster than a baseline decoder written in PYTHON, and that the distributed language model is very useful to improve translation quality in a large-scale task. We also describe an algorithm that exploits the *back-off* property of an m-gram model to maintain equivalent LM states, and show that better search quality is obtained with less search effort when the search space is organized to exploit this equivalence. We plan to incorporate some additional syntax-based components into the decoder and release it as an open-source toolkit.

Acknowledgments

We thank Philip Resnik, Chris Dyer, Smaranda Muresan and Adam Lopez for very helpful discussions, and the anonymous reviewers for their constructive comments. This research was partially supported by the Defense Advanced Research Projects Agency's GALE program via Contract N$\underline{o}$ HR0011-06-2-0001.

References

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2006. Large Language Models in Machine Translation. *In Proceedings of EMNLP 2007.*

David Chiang. 2006. An Introduction to Synchronous Grammars. Available at http://www.isi.edu/∼chiang/papers/synchtut.pdf.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201-228.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. *In Proceedings of ACL 2003.*

Ahmad Emami, Kishore Papineni, and Jeffrey Sorensen. 2007. Large-scale distributed language modeling. *In Proceedings of ICASSP 2007.*

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. *In Proceedings of COLING/ACL 2006.*

Liang Huang and David Chiang. 2005. Better k-best parsing. *In Proceedings of IWPT 2005.*

Liang Huang and David Chiang. 2007. Forest Rescoring: Faster Decoding with Integrated Language Models. *In Proceedings of the ACL 2007.*

Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. *In Proceedings of AMTA 2006.*

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. *In Proceedings of COLING-ACL 2006.*

Adam Lopez. 2007. Hierarchical Phrase-Based Translation with Suffix Arrays. *In Proceedings of EMNLP 2007.*

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. *In Proceedings of ACL 2003.*

Franz Josef Och and Hermann Ney. 2000. Improved statistical alignment models. *In Proceedings of ACL 2000.*

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. *In Proceedings of ACL 2002*.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency Treelet Translation: Syntactically Informed Phrasal SMT. *In Proceedings of ACL 2005*.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3-15.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. *In Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 901-904.

Ashish Venugopal, Andreas Zollmann, Stephan Vogel. 2007. An Efficient Two-Pass Approach to Synchronous-CFG Driven Statistical MT. *In Proceedings of NAACL 2007*.

Ying Zhang, Almut Silja Hildebrand, and Stephan Vogel. 2006. Distributed language modeling for n-best list reranking. *In Proceedings of EMNLP 2006*.