Hierarchical Phrase-Based Translation Representations

Gonzalo Iglesias[*] **Cyril Allauzen**[‡] **William Byrne**[*]
Adrià de Gispert[*] **Michael Riley**[‡]

[*]Department of Engineering, University of Cambridge, Cambridge, CB2 1PZ, U.K.
{gi212,wjb31,ad465}@eng.cam.ac.uk
[‡] Google Research, 76 Ninth Avenue, New York, NY 10011
{allauzen,riley}@google.com

Abstract

This paper compares several translation representations for a synchronous context-free grammar parse including CFGs/hypergraphs, finite-state automata (FSA), and pushdown automata (PDA). The representation choice is shown to determine the form and complexity of target LM intersection and shortest-path algorithms that follow. Intersection, shortest path, FSA expansion and RTN replacement algorithms are presented for PDAs. Chinese-to-English translation experiments using HiFST and HiPDT, FSA and PDA-based decoders, are presented using admissible (or exact) search, possible for HiFST with compact SCFG rulesets and HiPDT with compact LMs. For large rulesets with large LMs, we introduce a two-pass search strategy which we then analyze in terms of search errors and translation performance.

1 Introduction

Hierarchical phrase-based translation, using a *synchronous context-free translation grammar* (SCFG) together with an n-gram target language model (LM), is a popular approach in machine translation (Chiang, 2007). Given a SCFG G and an n-gram language model M, this paper focuses on how to *decode* with them, i.e. how to apply them to the source text to generate a target translation. Decoding has three basic steps, which we first describe in terms of the formal languages and relations involved, with data representations and algorithms to follow.

1. *Translating the source sentence s with G to give target translations:* $\mathcal{T} = \{s\} \circ \mathcal{G}$, a (weighted) context-free language resulting from the composition of a finite language and the algebraic relation $\mathcal{G}$ for SCFG G.
2. *Applying the language model to these target translations:* $\mathcal{L} = \mathcal{T} \cap \mathcal{M}$, a (weighted) context-free language resulting from the intersection of a context-free language and the regular language $\mathcal{M}$ for M.
3. *Searching for the translation and language model combination with the highest-probablity path:* $\hat{\mathcal{L}} = \text{argmax}_{l \in \mathcal{L}} L$

Of course, decoding requires explicit data representations and algorithms for combining and searching them. In common to the approaches we will consider here, s is applied to G by using the CYK algorithm in Step 1 and M is represented by a finite automaton in Step 2. The choice of the representation of $\mathcal{T}$ in many ways determines the remaining decoder representations and algorithms needed. Since $\{s\}$ is a finite language and we assume throughout that G does not allow unbounded insertions, $\mathcal{T}$ and $\mathcal{L}$ are, in fact, regular languages. As such, $\mathcal{T}$ and $\mathcal{L}$ have finite automaton representations T_f and L_f. In this case, weighted finite-state intersection and single-source shortest path algorithms (using negative log probabilities) can be used to solve Steps 2 and 3 (Mohri, 2009). This is the approach taken in (Iglesias et al., 2009a; de Gispert et al., 2010). Instead $\mathcal{T}$ and $\mathcal{L}$ can be represented by *hypergraphs* T_h and L_h (or very similarly context-free rules, and-or trees, or deductive systems). In this case, hypergraph intersection with a finite automaton and hypergraph shortest path algorithms can be used to solve Steps 2 and 3 (Huang, 2008). This is the approach taken by Chiang (2007). In this paper, we will consider another representation for context-free languages $\mathcal{T}$ and $\mathcal{L}$ as well, *pushdown automata* (PDA) T_p and L_p, familiar from formal

language theory (Aho and Ullman, 1972). We will describe PDA intersection with a finite automaton and PDA shortest-path algorithms in Section 2 that can be used to solve Steps 2 and 3. It cannot be over-emphasized that the CFG, hypergraph and PDA representations of $\mathcal{T}$ are used for their compactness rather than for expressing non-regular languages.

As presented so far, the search performed in Step 3 is *admissible* (or *exact*) – the true shortest path is found. However, the search space in MT can be quite large. Many systems employ aggressive pruning during the shortest-path computation with little theoretical or empirical guarantees of correctness. Further, such pruning can greatly complicate any complexity analysis of the underlying representations and algorithms. In this paper, we will exclude any inadmissible pruning in the shortest-path algorithm itself. This allows us in Section 3 to compare the computational complexity of using these different representations. We show that the PDA representation is particularly suited for decoding with large SCFGs and compact LMs.

We present Chinese-English translation results under the FSA and PDA translation representations. We describe a two-pass translation strategy which we have developed to allow use of the PDA representation in large-scale translation. In the first pass, translation is done using a lattice-generating version of the shortest path algorithm. The full translation grammar is used but with a compact, entropy-pruned version (Stolcke, 1998) of the full language model. This first-step uses admissible pruning and lattice generation under the compact language model. In the second pass, the original, unpruned LM is simply applied to the lattices produced in the first pass. We find that entropy-pruning and first-pass translation can be done so as to introduce very few search errors in the overall process; we can identify search errors in this experiment by comparison to exact translation under the full translation grammar and language model using the FSA representation. We then investigate a translation grammar which is large enough that exact translation under the FSA representation is not possible. We find that translation is possible using the two-pass strategy with the PDA translation representation and that gains in BLEU score result from using the larger translation grammar.

1.1 Related Work

There is extensive prior work on computational efficiency and algorithmic complexity in hierarchical phrase-based translation. The challenge is to find algorithms that can be made to work with large translation grammars and large language models.

Following the original algorithms and analysis of Chiang (2007), Huang and Chiang (2007) developed the cube-growing algorithm, and more recently Huang and Mi (2010) developed an incremental decoding approach that exploits left-to-right nature of the language models.

Search errors in hierarchical translation, and in translation more generally, have not been as extensively studied; this is undoubtedly due to the difficulties inherent in finding exact translations for use in comparison. Using a relatively simple phrase-based translation grammar, Iglesias et al. (2009b) compared search via cube-pruning to an exact FST implementation (Kumar et al., 2006) and found that cube-pruning suffered significant search errors. For Hiero translation, an extensive comparison of search errors between the cube pruning and FSA implementation was presented by Iglesias et al. (2009a) and de Gispert et al. (2010). Relaxation techniques have also recently been shown to finding exact solutions in parsing (Koo et al., 2010) and in SMT with tree-to-string translation grammars and trigram language models (Rush and Collins, 2011), much smaller models compared to the work presented in this paper.

Although entropy-pruned language models have been used to produce real-time translation systems (Prasad et al., 2007), we believe our use of entropy-pruned language models in two-pass translation to be novel. This is an approach that is widely-used in automatic speech recognition (Ljolje et al., 1999) and we note that it relies on efficient representation of very large search spaces $\mathcal{T}$ for subsequent rescoring, as is possible with FSAs and PDAs.

2 Pushdown Automata

In this section, we formally define pushdown automata and give intersection, shortest-path and related algorithms that will be needed later.

Informally, pushdown automata are finite automata that have been augmented with a stack. Typ-
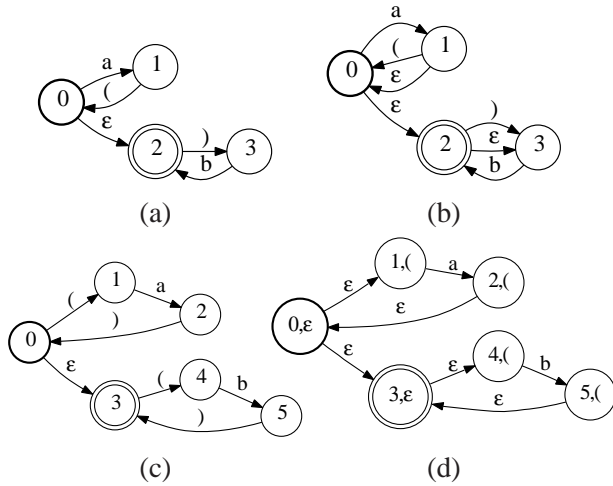
Figure 1: PDA Examples: (a) Non-regular PDA accepting $\{a^n b^n | n \in \mathbb{N}\}$. (b) Regular (but not bounded-stack) PDA accepting $a^* b^*$. (c) Bounded-stack PDA accepting $a^* b^*$ and (d) its expansion as an FSA.

ically this is done by adding a stack alphabet and labeling each transition with a stack operation (a stack symbol to be pushed onto, popped or read from the stack) in additon to the usual input label (Aho and Ullman, 1972; Berstel, 1979) and weight (Kuich and Salomaa, 1986; Petre and Salomaa, 2009). Our equivalent representation allows a transition to be labeled by a stack operation or a regular input symbol but not both. Stack operations are represented by pairs of open and close parentheses (pushing a symbol on and popping it from the stack). The advantage of this representation is that is identical to the finite automaton representation except that certain symbols (the parentheses) have special semantics. As such, several finite-state algorithms either immediately generalize to this PDA representation or do so with minimal changes. The algorithms described in this section have been implemented in the PDT extension (Allauzen and Riley, 2011) of the OpenFst library (Allauzen et al., 2007).

2.1 Definitions

A (restricted) Dyck language consist of "well-formed" or "balanced" strings over a finite number of pairs of parentheses. Thus the string ([( ) ( )] { } [ ]) ( ) is in the Dyck language over 3 pairs of parentheses.

More formally, let A and $\overline{A}$ be two finite alphabets such that there exists a bijection f from A to $\overline{A}$. Intuitively, f maps an open parenthesis to its corresponding close parenthesis. Let $\bar{a}$ denote $f(a)$ if $a \in A$ and $f^{-1}(a)$ if $a \in \overline{A}$. The *Dyck language* D_A over the alphabet $\widehat{A} = A \cup \overline{A}$ is then the language defined by the following context-free grammar: $S \to \epsilon$, $S \to SS$ and $S \to aS\bar{a}$ for all $a \in A$. We define the mapping $c_A : \widehat{A}^* \to \widehat{A}^*$ as follow. $c_A(x)$ is the string obtained by iteratively deleting from x all factors of the form $a\bar{a}$ with $a \in A$. Observe that $D_A = c_A^{-1}(\epsilon)$.

Let A and B be two finite alphabets such that $B \subseteq A$, we define the mapping $r_B : A^* \to B^*$ by $r_B(x_1 \ldots x_n) = y_1 \ldots y_n$ with $y_i = x_i$ if $x_i \in B$ and $y_i = \epsilon$ otherwise.

A *weighted pushdown automaton* (PDA) T over the tropical semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ is a 9-tuple $(\Sigma, \Pi, \overline{\Pi}, Q, E, I, F, \rho)$ where Σ is the finite input alphabet, Π and $\overline{\Pi}$ are the finite open and close parenthesis alphabets, Q is a finite set of states, $I \in Q$ the initial state, $F \subseteq Q$ the set of final states, $E \subseteq Q \times (\Sigma \cup \widehat{\Pi} \cup \{\epsilon\}) \times (\mathbb{R} \cup \{\infty\}) \times Q$ a finite set of transitions, and $\rho : F \to \mathbb{R} \cup \{\infty\}$ the final weight function. Let $e = (p[e], i[e], w[e], n[e])$ denote a transition in E.

A path π is a sequence of transitions $\pi = e_1 \ldots e_n$ such that $n[e_i] = p[e_{i+1}]$ for $1 \le i < n$. We then define $p[\pi] = p[e_1]$, $n[\pi] = n[e_n]$, $i[\pi] = i[e_1] \cdots i[e_n]$, and $w[\pi] = w[e_1] + \ldots + w[e_n]$.

A path π is accepting if $p[\pi] = I$ and $n[\pi] \in F$. A path π is balanced if $r_{\widehat{\Pi}}(i[\pi]) \in D_\Pi$. A balanced path π accepts the string $x \in \Sigma^*$ if it is a balanced accepting path such that $r_\Sigma(i[\pi]) = x$.

The *weight associated by* T to a string $x \in \Sigma^*$ is $T(x) = \min_{\pi \in P(x)} w[\pi] + \rho(n[\pi])$ where $P(x)$ denotes the set of balanced paths accepting x. A weighted language is recognizable by a weighted pushdown automaton iff it is context-free. We define the *size* of T as $|T| = |Q| + |E|$.

A PDA T has *a bounded stack* if there exists $K \in \mathbb{N}$ such that for any sub-path π of any balanced path in T: $|c_\Pi(r_{\widehat{\Pi}}(i[\pi]))| \le K$. If T has a bounded stack, then it represents a regular language. Figure 1 shows non-regular, regular and bounded-stack PDAs.

A *weighted finite automaton* (FSA) can be viewed as a PDA where the open and close parentheses alphabets are empty, see (Mohri, 2009) for a stand-alone definition.

2.2 Expansion Algorithm

Given a bounded-stack PDA T, the *expansion* of T is the FSA T' equivalent to T defined as follows.

A state in T' is a pair (q, z) where q is a state in T and $z \in \Pi^*$. A transition (q, a, w, q') in T results in a transition $((q, z), a', w, (q', z'))$ in T' only when: (a) $a \in \Sigma \cup \{\epsilon\}$, $z' = z$ and $a' = a$, (b) $a \in \Pi$, $z' = za$ and $a' = \epsilon$, or (c) $a \in \overline{\Pi}$, z' is such that $z = z'\overline{a}$ and $a' = \epsilon$. The initial state of T' is $I' = (I, \epsilon)$. A state (q, z) in T' is final if q is final in T and $z = \epsilon$ $(\rho'((q, \epsilon)) = \rho(q))$. The set of states of T' is the set of pairs (q, z) that can be reached from an initial state by transitions defined as above. The condition that T has a bounded stack ensures that this set is finite (since it implies that for any (q, z), $|z| \leq K$).

The complexity of the algorithm is linear in $O(|T'|) = O(e^{|T|})$. Figure 1d show the result of the algorithm when applied to the PDA of Figure 1c.

2.3 Intersection Algorithm

The class of weighted pushdown automata is closed under intersection with weighted finite automata (Bar-Hillel et al., 1964; Nederhof and Satta, 2003). Considering a pair (T_1, T_2) where one element is an FSA and the other element a PDA, then there exists a PDA $T_1 \cap T_2$, the *intersection* of T_1 and T_2, such that for all $x \in \Sigma^*$: $(T_1 \cap T_2)(x) = T_1(x) + T_2(x)$. We assume in the following that T_2 is an FSA. We also assume that T_2 has no input-ϵ transitions. When T_2 has input-ϵ transitions, an epsilon filter (Mohri, 2009; Allauzen et al., 2011) generalized to handle parentheses can be used.

A state in $T = T_1 \cap T_2$ is a pair (q_1, q_2) where q_1 is a state of T_1 and q_2 a state of T_2. The initial state is $I = (I_1, I_2)$. Given a transition $e_1 = (q_1, a, w_1, q_1')$ in T_1, transitions out of (q_1, q_2) in T are obtained using the following rules.

If $a \in \Sigma$, then e_1 can be matched with a transition (q_2, a, w_2, q_2') in T_2 resulting a transition $((q_1, q_2), a, w_1 + w_2, (q_1', q_2'))$ in T.

If $a = \epsilon$, then e_1 is matched with staying in q_2 resulting in a transition $((q_1, q_2), \epsilon, w_1, (q_1', q_2))$.

Finally, if $a \in \widehat{\Pi}$, e_1 is also matched with staying in q_2, resulting in a transition $((q_1, q_2), a, w_1, (q_1', q_2))$ in T.

A state (q_1, q_2) in T is final when both q_1 and q_2 are final, and then $\rho((q_1, q_2)) = \rho_1(q_1) + \rho_2(q_2)$.

SHORTESTDISTANCE(T)
1 **for** each $q \in Q$ and $a \in \Pi$ **do**
2      $B[q, a] \leftarrow \emptyset$
3 GETDISTANCE(T, I)
4 **return** $d[f, I]$

RELAX($q, s, w, \mathcal{S}$)
1 **if** $d[q, s] > w$ **then**
2      $d[q, s] \leftarrow w$
3   **if** $q \notin \mathcal{S}$ **then**
4   ENQUEUE($\mathcal{S}, q$)

GETDISTANCE(T, s)
1 **for** each $q \in Q$ **do**
2      $d[q, s] \leftarrow \infty$
3 $d[s, s] \leftarrow 0$
4 $\mathcal{S}_s \leftarrow s$
5 **while** $\mathcal{S}_s \neq \emptyset$ **do**
6      $q \leftarrow$ HEAD($\mathcal{S}_s$)
7   DEQUEUE($\mathcal{S}_s$)
8   **for** each $e \in E[q]$ **do**
9   **if** $i[e] \in \Sigma \cup \{\epsilon\}$ **then**
10   RELAX($n[e], s, d[q, s] + w[e], \mathcal{S}_s$)
11   **elseif** $i[e] \in \overline{\Pi}$ **then**
12              $B[s, \overline{i[e]}] \leftarrow B[s, \overline{i[e]}] \cup \{e\}$
13   **elseif** $i[e] \in \Pi$ **then**
14   **if** $d[n[e], n[e]]$ is undefined **then**
15   GETDISTANCE($T, n[e]$)
16   **for** each $e' \in B[n[e], i[e]]$ **do**
17                  $w \leftarrow d[q, s] + w[e] + d[p[e'], n[e]] + w[e']$
18   RELAX($n[e'], s, w, \mathcal{S}_s$)

Figure 2: PDA shortest distance algorithm. We assume that $F = \{f\}$ and $\rho(f) = 0$ to simplify the presentation.

The complexity of the algorithm is in $O(|T_1||T_2|)$.

2.4 Shortest Distance and Path Algorithms

A *shortest path* in a PDA T is a balanced accepting path with minimal weight and the *shortest distance* in T is the weight of such a path. We show that when T has a bounded stack, shortest distance and shortest path can be computed in $O(|T|^3 \log |T|)$ time (assuming T has no negative weights) and $O(|T|^2)$ space.

Given a state s in T with at least one incoming open parenthesis transition, we denote by C_s the set of states that can be reached from s by a balanced path. If s has several incoming open parenthesis transitions, a naive implementation might lead to the states in C_s to be visited up to exponentially many times. The basic idea of the algorithm is to memoize the shortest distance from s to states in C_s. The

pseudo-code is given in Figure 2.

GETDISTANCE(T, s) starts a new instance of the shortest-distance algorithm from s using the queue $\mathcal{S}_s$, initially containing s. While the queue is not empty, a state is dequeued and its outgoing transitions examined (line 5-9). Transitions labeled by non-parenthesis are treated as in Mohri (2009) (line 9-10). When the considered transition e is labeled by a close parenthesis, it is remembered that it balances all incoming open parentheses in s labeled by $\overline{i[e]}$ by adding e to $B[s, \overline{i[e]}]$ (line 11-12). Finally, when e is labeled with an open parenthesis, if its destination has not already been visited, a new instance is started from $n[e]$ (line 14-15). The destination states of all transitions balancing e are then relaxed (line 16-18).

The space complexity of the algorithm is quadratic for two reasons. First, the number of non-infinity $d[q, s]$ is $|Q|^2$. Second, the space required for storing B is at most in $O(|E|^2)$ since for each open parenthesis transition e, the size of $|B[n[e], i[e]]|$ is $O(|E|)$ in the worst case. This last observation also implies that the cumulated number of transitions examined at line 16 is in $O(N|Q| |E|^2)$ in the worst case, where N denotes the maximal number of times a state is inserted in the queue for a given call of GETDISTANCE. Assuming the cost of a queue operation is $\Gamma(n)$ for a queue containing n elements, the worst-case time complexity of the algorithm can then be expressed as $O(N|T|^3 \Gamma(|T|))$. When T contains no negative weights, using a shortest-first queue discipline leads to a time complexity in $O(|T|^3 \log |T|)$. When all the C_s's are acyclic, using a topological order queue discipline leads to a $O(|T|^3)$ time complexity.

In effect, we are solving a k-sources shortest-path problem with k single-source solutions. A potentially better approach might be to solve the k-sources or k-pairs problem directly (Hershberger et al., 2003).

When T has been obtained by converting an RTN or an hypergraph into a PDA (Section 2.5), the polynomial dependency in $|T|$ becomes a linear dependency both for the time and space complexities. Indeed, for each q in T, there exists a unique s such that $d[q, s]$ is non-infinity. Moreover, for each close parenthesis transisition e, there exists a unique open parenthesis transition e' such that $e \in B[n[e'], i[e']]$.

When each component of the RTN is acyclic, the complexity of the algorithm is hence in $O(|T|)$ in time and space.

The algorithm can be modified to compute the shortest path by keeping track of parent pointers.

2.5 Replacement Algorithm

A *recursive transition network* (RTN) can be specified by $(N, \Sigma, (T_\nu)_{\nu \in N}, S)$ where N is an alphabet of nonterminals, Σ is the input alphabet, $(T_\nu)_{\nu \in N}$ is a family of FSAs with input alphabet $\Sigma \cup N$, and $S \in N$ is the root nonterminal.

A string $x \in \Sigma^*$ is accepted by R if there exists an accepting path π in T_S such that recursively replacing any transition with input label $\nu \in N$ by an accepting path in T_ν leads to a path π^* with input x. The weight associated by R is the minimum over all such π^* of $w[\pi^*] + \rho_S(n[\pi^*])$.

Given an RTN R, the *replacement* of R is the PDA T equivalent to R defined by the 9-tuple $(\Sigma, \Pi, \overline{\Pi}, Q, E, I, F, \sigma, \rho)$ with $\Pi = Q = \bigcup_{\nu \in N} Q_\nu$, $I = I_S$, $F = F_S$, $\rho = \rho_S$, and $E = \bigcup_{\nu \in N} \bigcup_{e \in E_\nu} E^e$ where $E^e = \{e\}$ if $i[e] \notin N$ and $E^e = \{(p[e], n[e], w[e], I_\mu), (f, \overline{n[e]}, \rho_\mu(f), n[e]) | f \in F_\mu\}$ with $\mu = i[e] \in N$ otherwise.

The complexity of the construction is in $O(|T|)$. If $|F_\nu| = 1$, then $|T| = O(\sum_{\nu \in N} |T_\nu|) = O(|R|)$. Creating a superfinal state for each T_ν would lead to a T whose size is always linear in the size of R.

3 Hierarchical Phrase-Based Translation Representation

In this section, we compare several different representations for the target translations $\mathcal{T}$ of the source sentence s by synchronous CFG G prior to language model M application. As discussed in the introduction, $\mathcal{T}$ is a context-free language. For example, suppose it corresponds to:

$$S \to abXdg, \quad S \to acXfg, \quad \text{and } X \to bc.$$

Figure 3 shows several alternative representations of $\mathcal{T}$: Figure 3a shows the hypergraph representation of this grammar; there is a 1:1 correspondence between each production in the CFG and each hyperedge in the hypergraph. Figure 3b shows the RTN representation of this grammar with a 1:1 correspondence between each production in the CFG and each path in the RTN; this is the translation representation pro-
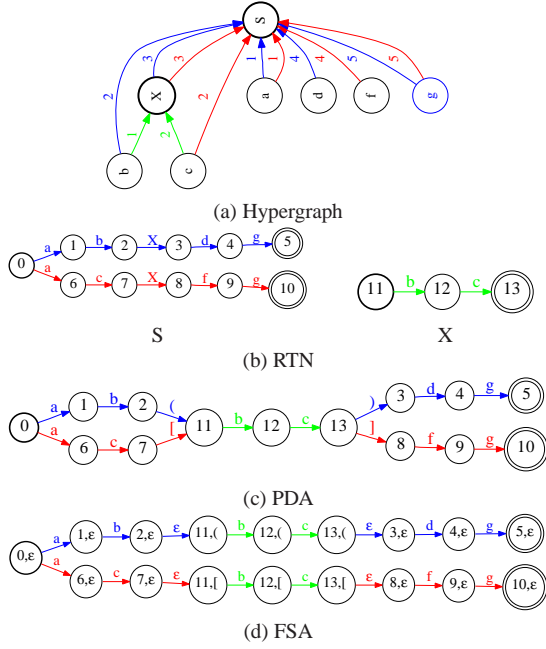
(a) Hypergraph

S   X

(b) RTN

(c) PDA

(d) FSA

Figure 3: Alternative translation representations

(a) RTN

(b) PDA

(c) FSA

Figure 4: Optimized translation representations

duced by the HiFST decoder (Iglesias et al., 2009a; de Gispert et al., 2010). Figure 3c shows the pushdown automaton representation generated from the RTN with the replacement algorithm of Section 2.5. Since s is a finite language and G does not allow unbounded insertion, T_p has a bounded stack and $\mathcal{T}$ is, in fact, a regular language. Figure 3d shows the finite-state automaton representation of $\mathcal{T}$ generated by the PDA using the expansion algorithm of Section 2.2. The HiFST decoder converts its RTN translation representation immediately into the finite-state representation using an algorithm equivalent to converting the RTN into a PDA followed by PDA expansion.

As shown in Figure 4, an advantage of the RTN, PDA, and FSA representations is that they can benefit from FSA epsilon removal, determinization and minimization algorithms applied to their components (for RTNs and PDAs) or their entirety (for FSAs). For the complexity discussion below, however, we disregard these optimizations. Instead we focus on the complexity of each MT step described in the introduction:

1. *SCFG Translation:* Assuming that the parsing of the input is performed by a CYK parse, then the CFG, hypergraph, RTN and PDA represen-
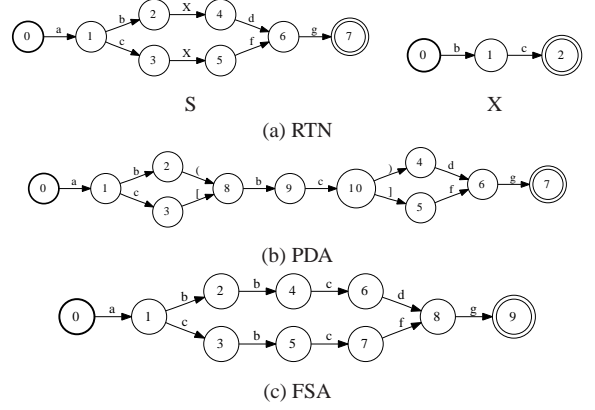
tations can be generated in $O(|s|^3|G|)$ time and space (Aho and Ullman, 1972). The FSA representation can require an additional $O(e^{|s|^3|G|})$ time and space since the PDA expansion can be exponential.

2. *Intersection:* The intersection of a CFG T_h with a finite automaton M can be performed by the classical Bar-Hillel algorithm (Bar-Hillel et al., 1964) with time and space complexity $O(|T_h||M|^3)$.[1] The PDA intersection algorithm from Section 2.3 has time and space complexity $O(|T_p||M|)$. Finally, the FSA intersection algorithm has time and space complexity $O(|T_f||M|)$ (Mohri, 2009).

3. *Shortest Path:* The shortest path algorithm on the hypergraph, RTN, and FSA representations requires linear time and space (given the underlying acyclicity) (Huang, 2008; Mohri, 2009). As presented in Section 2.4, the PDA representation can require time cubic and space quadratic in $|M|$.[2]

Table 1 summarizes the complexity results. Note the PDA representation is equivalent in time and superior in space to the CFG/hypergraph representation, in general, and it can be superior in both space

---

[1] The modified Bar-Hillel construction described by Chiang (2007) has time and space complexity $O(|T_h||M|^4)$; the modifications were introduced presumably to benefit the subsequent pruning method employed (but see Huang et al. (2005)).

[2] The time (resp. space) complexity is not cubic (resp. quadratic) in $|T_p||M|$. Given a state q in T_p, there exists a unique s_q such that q belongs to C_{s_q}. Given a state (q_1, q_2) in $T_p \cap M$, $(q_1, q_2) \in C_{(s_1, s_2)}$ only if $s_1 = s_{q_1}$, and hence (q_1, q_2) belongs to at most $|M|$ components.

Representation	Time Complexity	Space Complexity
CFG/hypergraph	$O(|s|^3 |G| |M|^3)$	$O(|s|^3 |G| |M|^3)$
PDA	$O(|s|^3 |G| |M|^3)$	$O(|s|^3 |G| |M|^2)$
FSA	$O(e^{|s|^3 |G|} |M|)$	$O(e^{|s|^3 |G|} |M|)$

Table 1: Complexity using various target translation representations.

0	7.5×10^{-9}	7.5×10^{-8}	7.5×10^{-7}
207.5	20.2	4.1	0.9

Table 2: Number of ngrams (in millions) in the 1st pass 4-gram language models obtained with different θ values (top row).

and time to the FSA representation depending on the relative SCFG and LM sizes. The FSA representation favors smaller target translation sets and larger language models. Should a better complexity PDA shortest path algorithm be found, this conclusion could change. In practice, the PDA and FSA representations benefit hugely from the optimizations mentioned above, these optimizations improve the time and space usage by one order of magnitude.

4 Experimental Framework

We use two hierarchical phrase-based SMT decoders. The first one is a lattice-based decoder implemented with weighted finite-state transducers (de Gispert et al., 2010) and described in Section 3. The second decoder is a modified version using PDAs as described in Section 2. In order to distinguish both decoders we call them HiFST and HiPDT, respectively. The principal difference between the two decoders is where the finite-state *expansion* step is done. In HiFST, the RTN representation is immediately expanded to an FSA. In HiPDT, this expansion is delayed as late as possible - in the output of the shortest path algorithm. Another possible configuration is to expand after the LM intersection step but before the shortest path algorithm; in practice this is quite similar to HiFST.

In the following sections we report experiments in Chinese-to-English translation. For translation model training, we use a subset of the GALE 2008 evaluation parallel text;[3] this is 2.1M sentences and approximately 45M words per language. We report translation results on a development set *tune-nw* (1,755 sentences) and a test set *test-nw* (1,671 sentences). These contain translations produced by the GALE program and portions of the newswire sections of MT02 through MT06. In tuning the sys-

tems, standard MERT (Och, 2003) iterative parameter estimation under IBM BLEU[4] is performed on the development set.

The parallel corpus is aligned using MTTK (Deng and Byrne, 2008) in both source-to-target and target-to-source directions. We then follow standard heuristics (Chiang, 2007) and filtering strategies (Iglesias et al., 2009b) to extract hierarchical phrases from the union of the directional word alignments. We call a translation grammar the set of rules extracted from this process. We extract two translation grammars:

- A restricted grammar where we apply the following additional constraint: rules are only considered if they have a forward translation probability $p > 0.01$. We call this G_1. As will be discussed later, the interest of this grammar is that decoding under it can be exact, that is, without any pruning in search.
- An unrestricted one without the previous constraint. We call this G_2. This is a superset of the previous grammar, and exact search under it is not feasible for HiFST: pruning is required in search.

The initial English language model is a Kneser-Ney 4-gram estimated over the target side of the parallel text and the AFP and Xinhua portions of monolingual data from the English Gigaword Fourth Edition (LDC2009T13). This is a total of 1.3B words. We will call this language model M_1. For large language model rescoring we also use the LM M_2 obtained by interpolating M_1 with a zero-cutoff stupid-backoff (Brants et al., 2007) 5-gram estimated using 6.6B words of English newswire text.

We next describe how we build translation systems using entropy-pruned language models.

1. We build a baseline HiFST system that uses M_1 and a hierarchical grammar G, parameters being optimized with MERT under BLEU.

---

[3]See http://projects.ldc.upenn.edu/gale/data/catalog.html. We excluded the UN material and the LDC2002E18, LDC2004T08, LDC2007E08 and CUDonga collections.

[4]See ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13.pl

2. We then use entropy-based pruning of the language model (Stolcke, 1998) under a relative perplexity threshold of θ to reduce the size of M_1. We will call the resulting language model as M_1^θ. Table 2 shows the number of n-grams (in millions) obtained for different θ values.

3. We translate with M_1^θ using the same parameters obtained in MERT in step 1, except for the word penalty, tuned over the lattices under BLEU performance. This produces a translation lattice in the topmost cell that contains hypotheses with exact scores under the translation grammar and M_1^θ.

4. Translation lattices in the topmost cell are pruned with a likelihood-based beam width β.

5. We remove the M_1^θ scores from the pruned translation lattices and reapply M_1, moving the word penalty back to the original value obtained in MERT. These operations can be carried out efficiently via standard FSA operations.

6. Additionally, we can rescore the translation lattices obtained in steps 1 or 5 with the larger language model M_2. Again, this can be done via standard FSA operations.

Note that if $\beta = \infty$ or if $\theta = 0$, the translation lattices obtained in step 1 should be identical to the ones of step 5. While the goal is to increase θ to reduce the size of the language model used at Step 3, β will have to increase accordingly so as to avoid pruning away desirable hypotheses in Step 4. If β defines a sufficiently wide beam to contain the hypotheses which would be favoured by M_1, faster decoding with M_1^θ would be possible without incurring search errors M_1. This is investigated next.

5 Entropy-Pruned LM in Rescoring

In Table 3 we show translation performance under grammar G_1 for different values of θ. Performance is reported after first-pass decoding with M_1^θ (see step 3 in Section 4), after rescoring with M_1 (see step 5) and after rescoring with M_2 (see step 6). The baseline (experiment number 1) uses $\theta = 0$ (that is, M_1) for decoding.

Under translation grammar G_1, HiFST is able to generate an FSA with the entire space of possible candidate hypotheses. Therefore, any degradation

in performance is only due to the M_1^θ involved in decoding and the β applied prior to rescoring.

As shown in row number 2, for $\theta \le 10^{-9}$ the system provides the same performance to the baseline when $\beta > 8$, while decoding time is reduced by roughly 40%. This is because M_1^θ is 10% of the size of the original language model M_1, as shown in Table 2. As M_1^θ is further reduced by increasing θ (see rows number 3 and 4), decoding time is also reduced. However, the beam width β required in order to recover the good hypotheses in rescoring increases, reaching 12 for experiment 3 and 15 for experiment 4.

Regarding rescoring with the larger M_2 (step 6 in Section 4), the system is also able to match the baseline performance as long as β is wide enough, given the particular M_1^θ used in first-pass decoding. Interestingly, results show that a similar β value is needed when rescoring either with M_1 or M_2.

The usage of entropy-pruned language models increments speed at the risk of search errors. For instance, comparing the outputs of systems 1 and 2 with $\beta = 10$ in Table 3 we find 45 different 1-best hypotheses, even though the BLEU score is identical. In other words, we have 45 cases in which system 2 is not able to recover the baseline output because the 1st-pass likelihood beam β is not wide enough. Similarly, system 3 fails in 101 cases ($\beta = 12$) and system 4 fails in 95 cases. Interestingly, some of these sentences would require impractically huge beams. This might be due to the Kneser-Ney smoothing, which interacts badly with entropy pruning (Chelba et al., 2010).

6 Hiero with PDAs and FSAs

In this section we contrast HiFST with HiPDT under the same translation grammar and entropy-pruned language models. Under the constrained grammar G_1 their performance is identical as both decoders can generate the entire search space which can then be rescored with M_1 or M_2 as shown in the previous section.

Therefore, we now focus on the unconstrained grammar G_2, where exact search is not feasible for HiFST. In order to evaluate this problem, we run both decoders over *tune-nw*, restricting memory usage to 10 gigabytes. If this limit is reached in decod-

HiFST ($G_1 + M_1^\theta$)						+M_1		+M_2	
#	θ	*tune-nw*	*test-nw*	*time*	β	*tune-nw*	*test-nw*	*tune-nw*	*test-nw*
1	**0** (M_1)	**34.3**	**34.5**	**0.68**	-	-	-	**34.8**	**35.6**
2	7.5×10^{-9}	32.0	32.8	0.38	*10*	34.3	34.5	34.8	35.6
					9			34.9	35.5
					8				
3	7.5×10^{-8}	29.5	30.0	0.28	*12*	34.2	34.5	34.7	35.6
					9	34.3	34.4	34.8	35.2
					8	34.2			35.1
4	7.5×10^{-7}	26.0	26.4	0.20	*15*	34.2	34.5	34.7	35.6
					12		34.4		35.5

Table 3: Results (lowercase IBM BLEU scores) under G_1 with various M_1^θ as obtained with several values of θ. Performance in subsequent rescoring with M_1 and M_2 after likelihood-based pruning of the translation lattices for various β is also reported. Decoding time, in seconds/word over *test-nw*, refers strictly to first-pass decoding.

Exact search for $G_2 + M_1^\theta$ with memory usage under 10 GB							
#	θ	HiFST			HiPDT		
		Success	**Failure**		**Success**	**Failure**	
			Expand	*Compose*		*Compose*	*Expand*
2	7.5×10^{-9}	12	51	37	40	8	52
3	7.5×10^{-8}	16	53	31	76	1	23
4	7.5×10^{-7}	18	53	29	99.8	0	0.2

Table 4: Percentage of success in producing the 1-best translation under G_2 with various M_1^θ when applying a hard memory limitation of 10 GB, as measured over *tune-nw* (1755 sentences). If decoder fails, we report what step was being done when the limit was reached. HiFST could be expanding into an FSA or composing the FSA with M_1^θ; HiPDT could be PDA composing with M_1^θ or PDA expanding into an FSA.

HiPDT ($G_2 + M_1^\theta$)				+M_1		+M_2	
θ	*tune-nw*	*test-nw*	β	*tune-nw*	*test-nw*	*tune-nw*	*test-nw*
7.5×10^{-7}	25.7	26.3	15	34.6	34.8	35.2	36.1

Table 5: HiPDT performance on grammar G_2 with $\theta = 7.5 \times 10^{-7}$. Exact search with HiFST is not possible under these conditions: pruning during search would be required.

ing, the process is killed[5]. We report what internal decoding operation caused the system to crash. For HiFST, these include expansion into an FSA (*Expand*) and subsequent intersection with the language model (*Compose*). For HiPDT, these include PDA intersection with the language model (*Compose*) and subsequent expansion into an FSA (*Expand*), using algorithms described in Section 2.

Table 4 shows the number of times each decoder succeeds in finding a hypothesis given the memory limit, and the operations being carried out when they fail to do so, when decoding with various M_1^θ. With $\theta = 7.5 \times 10^{-9}$ (row 2), HiFST can only decode 218 sentences, while HiPDT succeeds in 703 cases. The

differences between both decoders increase as the M_1^θ is more reduced, and for $\theta = 7.5 \times 10^{-7}$ (row 4), HiPDT is able to perform exact search over all but three sentences.

Table 5 shows performance using the latter configuration (Table 4, row 4). After large language model rescoring, HiPDT improves 0.5 BLEU over baseline with G_1 (Table 3, row 1).

7 Discussion and Conclusion

HiFST fails to decode mainly because the expansion into an FST leads to far too big search spaces (e.g. fails 938 times under $\theta = 7.5 \times 10^{-8}$). If it succeeds in expanding the search space into an FST, the decoder still has to compose with the language model, which is also critical in terms of memory us-

---
[5]We used *ulimit* command. The experiment was carried out over machines with different configurations and load. Therefore, these numbers must be considered as approximate values.

age (fails 536 times). In contrast, HiPDT creates a PDA, which is a more compact representation of the search space and allows efficient intersection with the language model before expansion into an FST. Therefore, the memory usage is considerably lower. Nevertheless, the complexity of the language model is critical for the PDA intersection and very specially the PDA expansion into an FST (fails 403 times for $\theta = 7.5 \times 10^{-8}$).

With the algorithms presented in this paper, decoding with PDAs is possible for any translation grammar as long as an entropy pruned LM is used. While this allows exact decoding, it comes at the cost of making decisions based on less complex LMs, although this has been shown to be an adequate strategy when applying compact CFG rulesets. On the other hand, HiFST cannot decode under large translation grammars, thus requiring pruning during lattice construction, but it can apply an unpruned LM in this process. We find that with carefully designed pruning strategies, HiFST can match the performance of HiPDT reported in Table 5. But without pruning in search, expansion directly into an FST would lead to an explosion in terms of memory usage. Of course, without memory constraints both strategies would reach the same performance.

Overall, these results suggest that HiPDT is more robust than HiFST when using complex hierarchical grammars. Conversely, FSTs might be more efficient for search spaces described by more constrained hierarchical grammars. This suggests that a hybrid solution could be effective: we could use PDAs or FSTs e.g. depending on the number of states of the FST representing the expanded search space, or other conditions.

8 Acknowledgments

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1-2. Prentice-Hall.

Cyril Allauzen and Michael Riley, 2011. *Pushdown Transducers*. http://pdt.openfst.org.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of CIAA*, pages 11–23. http://www.openfst.org.

Cyril Allauzen, Michael Riley, and Johan Schalkwyk. 2011. Filters for efficient composition of weighted finite-state transducers. In *Proceedings of CIAA*, volume 6482 of *LNCS*, pages 28–38. Springer.

Y. Bar-Hillel, M. Perles, and E. Shamir. 1964. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, pages 116–150. Addison-Wesley.

Jean Berstel. 1979. *Transductions and Context-Free Languages*. Teubner.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of EMNLP-ACL*, pages 858–867.

Ciprian Chelba, Thorsten Brants, Will Neveitt, and Peng Xu. 2010. Study on interaction between entropy pruning and kneser-ney smoothing. In *Proceedings of Interspeech*, pages 2242–2245.

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

Adrià de Gispert, Gonzalo Iglesias, Graeme Blackwood, Eduardo R. Banga, and William Byrne. 2010. Hierarchical phrase-based translation with weighted finite state transducers and shallow-n grammars. *Computational Linguistics*, 36(3).

Yonggang Deng and William Byrne. 2008. HMM word and phrase alignment for statistical machine translation. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(3):494–507.

Manfred Drosde, Werner Kuick, and Heiko Vogler, editors. 2009. *Handbook of Weighted Automata*. Springer.

John Hershberger, Subhash Suri, and Amit Bhosle. 2003. On the difficulty of some shortest path problems. In *Proceedings of STACS*, volume 2607 of *LNCS*, pages 343–354. Springer.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of ACL*, pages 144–151.

Liang Huang and Haitao Mi. 2010. Efficient incremental decoding for tree-to-string translation. In *Proceedings of EMNLP*, pages 273–283.

Liang Huang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 65–73, Stroudsburg, PA, USA. Association for Computational Linguistics.

Liang Huang. 2008. Advanced dynamic programming in semiring and hypergraph frameworks. In *Proceedings of COLING*, pages 1–18.

Gonzalo Iglesias, Adrià de Gispert, Eduardo R. Banga, and William Byrne. 2009a. Hierarchical phrase-based translation with weighted finite state transducers. In *Proceedings of NAACL-HLT*, pages 433–441.

Gonzalo Iglesias, Adrià de Gispert, Eduardo R. Banga, and William Byrne. 2009b. Rule filtering by pattern for efficient hierarchical translation. In *Proceedings of EACL*, pages 380–388.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pages 1288–1298.

Werner Kuich and Arto Salomaa. 1986. *Semirings, automata, languages*. Springer.

Shankar Kumar, Yonggang Deng, and William Byrne. 2006. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12(1):35–75.

Andrej Ljolje, Fernando Pereira, and Michael Riley. 1999. Efficient general lattice generation and rescoring. In *Proceedings of Eurospeech*, pages 1251–1254.

Mehryar Mohri. 2009. Weighted automata algorithms. In Drosde et al. (Drosde et al., 2009), chapter 6, pages 213–254.

Mark-Jan Nederhof and Giorgio Satta. 2003. Probabilistic parsing as intersection. In *Proceedings of 8th International Workshop on Parsing Technologies*, pages 137–148.

Franz J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Ion Petre and Arto Salomaa. 2009. Algebraic systems and pushdown automata. In Drosde et al. (Drosde et al., 2009), chapter 7, pages 257–289.

R. Prasad, K. Krstovski, F. Choi, S. Saleem, P. Natarajan, M. Decerbo, and D. Stallard. 2007. Real-time speech-to-speech translation for pdas. In *Proceedings of IEEE International Conference on Portable Information Devices*, pages 1–5.

Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of ACL-HLT*, pages 72–82.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proceedings of DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274.