Pushdown Automata in Statistical Machine Translation

Cyril Allauzen[*]
Google Research

Bill Byrne[**]
University of Cambridge

Adrià de Gispert[**]
University of Cambridge

Gonzalo Iglesias[**]
University of Cambridge

Michael Riley[*]
Google Research

This paper describes the use of pushdown automata (PDA) in the context of statistical machine translation and alignment under a synchronous context-free grammar. We use PDAs to compactly represent the space of candidate translations generated by the grammar when applied to an input sentence. General-purpose PDA algorithms for replacement, composition, shortest path, and expansion are presented. We describe HiPDT, a hierarchical phrase-based decoder using the PDA representation and these algorithms. We contrast the complexity of this decoder with a decoder based on a finite state automata (FSA) representation, showing that PDAs provide a more suitable framework to achieve exact decoding for larger SCFGs and smaller language models. We assess this experimentally on a large-scale Chinese-to-English alignment and translation task. In translation, we propose a two-pass decoding strategy involving a weaker language model in the first-pass to address the results of PDA complexity analysis. We study in depth the experimental conditions and tradeoffs in which HiPDT can achieve state-of-the-art performance for large-scale SMT.

1. Introduction

Synchronous context-free grammars (SCFGs) are now widely used in statistical machine translation, with Hiero as the preeminent example (Chiang 2007). Given an SCFG and an n-gram language model, the challenge is to *decode* with them, i.e. to apply them to source text to generate a target translation.

Decoding is complex in practice, but it can be described simply and exactly in terms of the formal languages and relations involved. We will use use this description to introduce and analyze pushdown automata (PDAs) for machine translation. This formal description will allow close comparison of PDAs to existing decoders which are based on other forms of automata. Decoding can be described in terms of the following steps:

[*] Google Research, 76 Ninth Avenue, New York, NY 10011. E-mail: {allauzen,riley}@google.com
[**] University of Cambridge, Department of Engineering, CB2 1PZ Cambridge, U.K.
and SDL, Wellington House, East Road, Cambridge, UK CB1 1BH. E-mail:{wjb31,ad465,gi212}@eng.cam.ac.uk

1. *Translation:* $\mathcal{T} = \Pi_2(\{s\} \circ \mathcal{G})$
   The first step is to compose the finite language $\{s\}$, which represents the source sentence to be translated, with the the algebraic relation $\mathcal{G}$ for the translation grammar G. The result of this composition projected on the output side is $\mathcal{T}$, a weighted context-free grammar that contains all possible translations of s under G. Following the usual definition of Hiero grammars, we assume that G does not allow unbounded insertions so that $\mathcal{T}$ is a regular language.
2. *Language Model Application:* $\mathcal{L} = \mathcal{T} \cap \mathcal{M}$
   The next step is to compose the result of Step 1 with the weighted regular grammar $\mathcal{M}$ defined by the n-gram language model, M. The result of this composition is $\mathcal{L}$, whose paths are weighted by the combined language model and translation scores.
3. *Search:* $\hat{l} = \mathrm{argmax}_{l \in \mathcal{L}} L$
   The final step is to find the path through $\mathcal{L}$ which has the best combined translation and language model score.

The composition $\{s\} \circ \mathcal{G}$ in Step 1 that generates $\mathcal{T}$ can be performed by a modified CYK algorithm (Chiang 2007). Our interest is in the different types of automata that can be used to represent $\mathcal{T}$ as it is produced by this composition. We focus on three types of representations: hypergraphs (Chiang 2007), WFSAs (Iglesias et al. 2009a; de Gispert et al. 2010), and PDAs. We will give a formal definition of PDAs in Section 2, but we will first illustrate and compare these different representations by a simple example.

Consider translating a source sentence '$s_1\ s_2\ s_3$' with a simple Hiero grammar G :

$$X \rightarrow \langle s_1,\ t_2\ t_3 \rangle$$
$$S \rightarrow \langle X\ s_2\ s_3,\ t_1\ t_2\ X\ t_4\ t_7 \rangle$$
$$S \rightarrow \langle X\ s_2\ s_3,\ t_1\ t_3\ X\ t_6\ t_7 \rangle$$

Step 1 yields the translations $\mathcal{T} = \{$'$t_1\ t_2\ t_2\ t_3\ t_4\ t_7$', '$t_1\ t_3\ t_2\ t_3\ t_6\ t_7$'$\}$, and Figure 1 gives examples of the different representations of these translations. We summarize the salient features of these representations as they are used in decoding.

Hypergraphs. As described by Chiang (2007), a Hiero decoder can generate translations in form of a hypergraph, as in Figure 1a. As the figure shows, there is a 1-to-1 correspondence between each production in the CFG and each hyperedge in the hypergraph.
   Decoding proceeds by intersecting the translation hypergraph with a language model, represented as a finite automaton, yielding $\mathcal{L}$ as a hypergraph. Step 3 yields a translation by finding the shortest path through the hypergraph $\mathcal{L}$ (Huang 2008).

Weighted Finite State Automata (WFSAs). Because $\mathcal{T}$ is a regular language and M is represented by a finite automaton, it follows that $\mathcal{T}$ and $\mathcal{L}$ can themselves be represented as finite automata. Consequently steps 2 and 3 can be solved using weighted finite-state intersection and single-source shortest path algorithms, respectively (Mohri 2009). This is the general approach adopted in the HiFST decoder (Iglesias et al. 2009a; de Gispert et al. 2010), which first represents $\mathcal{T}$ as a Recursive Transition Network (RTN) and then performs expansion to generate a WFSA.
   Figure 1b shows the space of translations for this example represented as an RTN. Like the hypergraph, it also has a 1:1 correspondence between each production in the CFG and paths in the RTN components. The recursive RTN itself can be expanded into a single WFSA, as shown in Figure 1c. Intersection and shortest path algorithms are available for both of these WFSAs.

(a) Hypergraph

(b) RTN

(c) FSA

(d) PDA
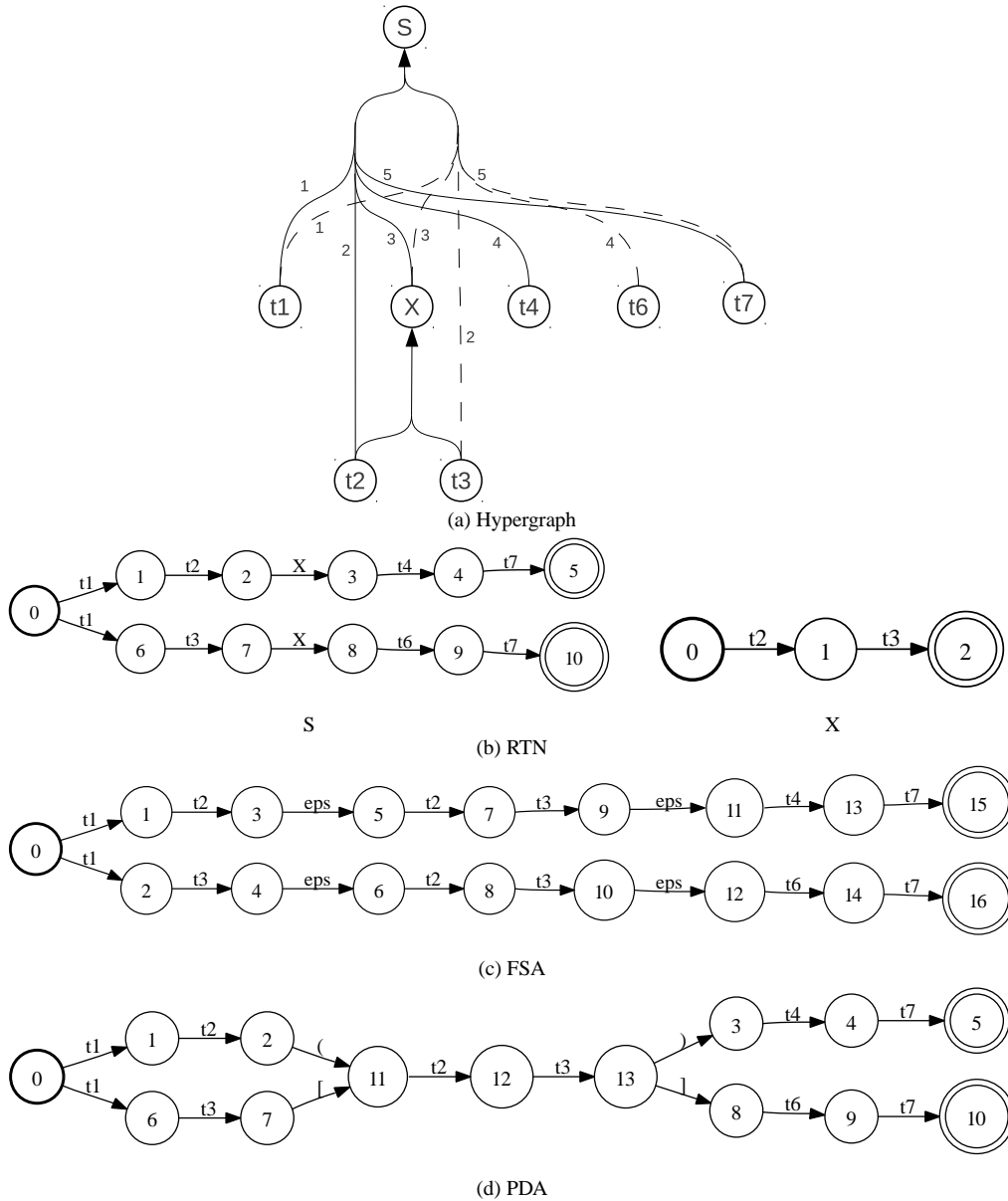
Figure 1
Alternative representations of the regular language of possible translation candidates. Valid paths through the PDA must have balanced parentheses.

Pushdown Automata. Like WFSAs, PDAs are easily generated from RTNs, as will be described later, and Figure 1d gives the PDA representation for this example. The PDA represents the same language as the FSA, but with fewer states. Procedures to carry out steps 2 and 3 in decoding will be described in subsequent sections.

We will show that PDAs provide a general framework to describe key aspects of several existing and novel translation algorithms. We note that PDAs have long been used to describe
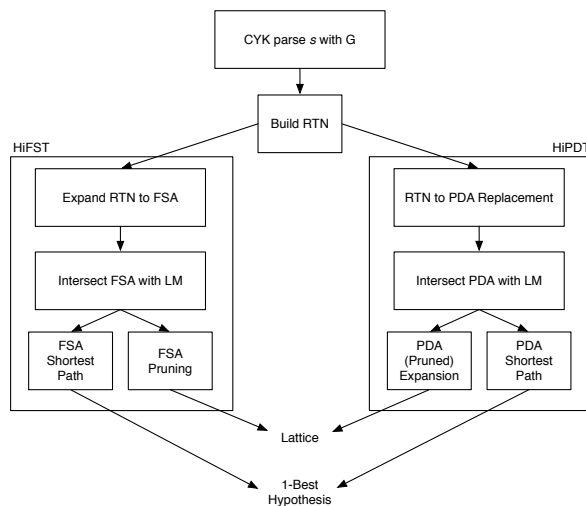
Figure 2
HiPDT versus HiFST: general flow and high level operations.

parsing algorithms (Aho and Ullman 1972; Lang 1974), and it is well known that *pushdown transducers*, the extended version of PDA with input and output labels in each transition, do not have the expressive power needed to generate synchronous context-free languages. For this reason, we do not use PDAs to implement Step 1 in decoding: throughout this paper a CYK-like parsing algorithm is always used for Step 1. However we do use PDAs to represent the regular languages produced in Step 1 and in the intersection and shortest distance operations needed for Steps 2 and 3.

1.1 HiPDT: hierarchical phrase-based translation with PDAs

We introduce HiPDT, a hierarchical phrase-based decoder that uses a PDA representation for the target language. The architecture of the system is shown in Figure 2, where we contrast it to HiFST (de Gispert et al. 2010). Both decoders parse the sentence with a grammar G using a modified version of the CYK algorithm to generate the translation search space as an RTN. Each decoder then follows a different path: HiFST expands the RTN into an FSA, intersects it with the language model and then prunes the result; HiPDT performs the following steps:

1. Convert the RTN into PDA using the *replacement* algorithm. The PDA representation for the example grammar in Section 1 is shown in Figure 1. The algorithm will be described in Section 3.2.
2. Apply the language model scores to the PDA by *composition*. This operation is described in Section 3.3.
3. Perform either one of the following operations:
   (a) *Shortest path* through the PDA to get the exact best translation under the model. Shortest distance/path algorithm is described in Section 3.4.
   (b) *Pruned Expansion* to an FSA. This expansion uses admissible pruning and outputs a lattice. We do this for posterior rescoring steps. The algorithm will be presented in detail in Sections 3.5 and 3.5.2.

The principal difference between the two decoders is the point at which finite-state expansion is performed. In HiFST, the RTN representation is immediately expanded to an FSA. In HiPDT, the PDA pruned expansion or shortest path computation is done after the language model is applied, so that all computation is done with respect to both the translation and language model scores.

The use of RTNs as an initial translation representation is somewhat influenced by the development history of our FST and SMT systems. RTN algorithms were available in OpenFST at the time HiFST was developed. HiPDT was developed as an extension to HiFST using PDA algorithms, and these have subsequently been included in OpenFST. A possible alternative approach could be to produce a PDA directly by traversing the CYK grid. WFSAs could then be generated by PDA expansion, with a computational complexity in speed and memory usage similar to the RTN-based approach. We present RTNs as the initial translation representation because the generation of RTNs during parsing is straightforward and has been previously presented (de Gispert et al. 2010). We note however that RTN composition is algorithmically more complex than PDA (and FSA) composition, so that RTNs themselves are not ideal representations of $\mathcal{T}$ if a language model is to be applied. Composition of PDAs with FSAs will be discussed in Section 3.3.

Figure 3 continues the simple translation example from earlier, showing how HiPDT and HiFST both benefit from the compactness offered by WFSA epsilon removal, determinization and minimization operations. When applied to PDAs, these operations treat parentheses as regular symbols. Compact representations of RTNs are shared by both approaches. Figure 4 illustrates the PDA representation of the translation space under a slightly more complex grammar that includes rules with alternative orderings of nonterminals. The rule $S \rightarrow \langle X_1\ s_2\ X_2,\ t_1\ X_1\ X_2 \rangle$ produces the sequence '$t_1\ t_3\ t_4\ t_5\ t_6$', while $S \rightarrow \langle X_1\ s_2\ X_2,\ t_2\ X_2\ X_1 \rangle$ produces '$t_2\ t_5\ t_6\ t_3\ t_4$'. The PDA efficiently represents the alternative orderings of the phrases '$t_3\ t_4$' and '$t_5\ t_6$' allowed under this grammar.

In addition to translation, this architecture can also be used directly to carry out source-to-target alignment, or synchronous parsing, under the SCFG in a two-step composition rather than one synchronous parsing stage. For example, by using M as the automata which accepts '$t_1\ t_2\ t_3\ t_6\ t_7$', Step 2 will yield all derivations that yield this string as a translation of the source string. This is the approach taken in (Iglesias et al. 2009a; de Gispert et al. 2010) for the RTN/FSA and in (Dyer 2010b) for hypergraphs. In Section 4 we analyze how PDAs can be used for alignment.

1.2 Goals

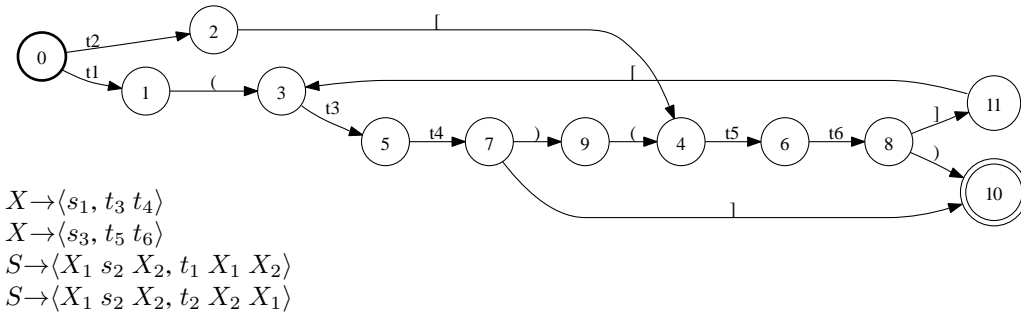We summarize here the aims of this paper.

We will show how PDAs can be used as compact representations of the space $\mathcal{T}$ of candidate translations generated by a hierarchical phrase-based SCFG when applied to an input sentence s and intersected with a language model M.
We have described the architecture of HiPDT, a hierarchical phrase-based decoder based on PDAs, and have identified the general-purpose algorithms needed to perform translation and alignment; in doing so we have highlighted the similarities and differences relative to translation with FSAs (Section 1.1). We will provide a formal description of PDAs (Section 2) and present in detail the associated PDA algorithms required to carry out Steps 2 and 3 above, including RTN replacement, composition, shortest path, expansion and pruned expansion (Section 3).

We will show both theoretically and experimentally that the PDA representation is well suited for exact decoding under a large SCFG and a small language model.

(a) Optimized RTN

(b) Optimized FSA
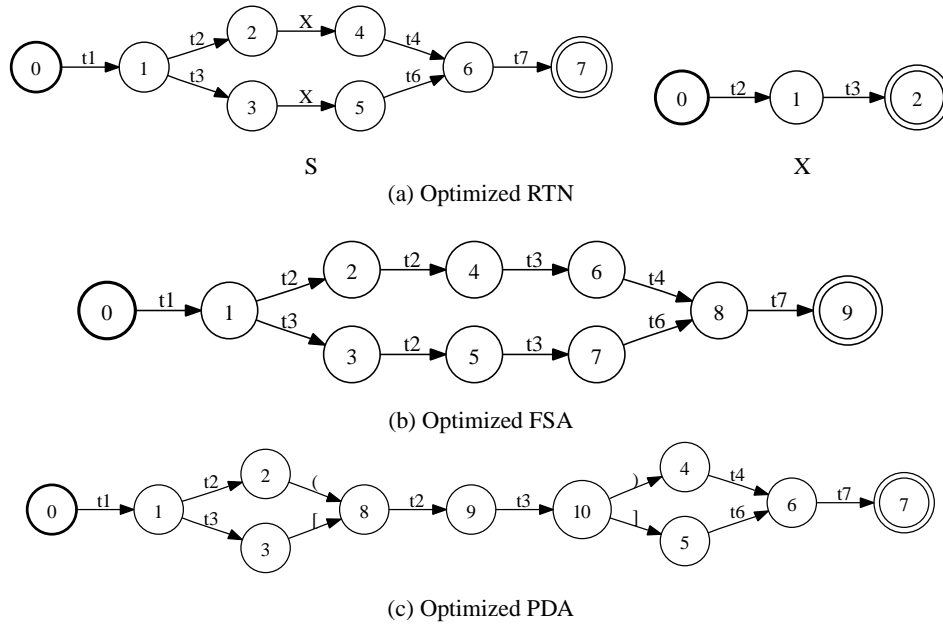
(c) Optimized PDA

Figure 3
Optimized representations of the regular language of possible translation candidates.

$X \rightarrow \langle s_1, t_3\ t_4 \rangle$
$X \rightarrow \langle s_3, t_5\ t_6 \rangle$
$S \rightarrow \langle X_1\ s_2\ X_2, t_1\ X_1\ X_2 \rangle$
$S \rightarrow \langle X_1\ s_2\ X_2, t_2\ X_2\ X_1 \rangle$

Figure 4
Example of translation grammar with reordered nonterminals and the PDA representing the result of
applying the grammar to input sentence $s_1\ s_2\ s_3$.

An analysis of decoder complexity in terms of the automata used in the representation
is presented (Section 3). One important aspect of the translation task is whether the
search for the best translation is *admissible* (or *exact*) under the translation and language
models. Stated differently, we wish to know whether a decoder produces the actual shortest
path found or whether some form of pruning might have introduced search errors. In
our formulation, we can exclude inadmissible pruning from the shortest-path algorithms,
and doing so makes it straightforward to compare the computational complexity of a
full translation pipeline using different representations of $\mathcal{T}$ (Section 4). We empirically
demonstrate that a PDA representation is superior to an FSA representation in the ability
to perform exact decoding both in an ITG-style word alignment task and in a translation

task with a small language model (Section 4). In these experiments we take HiFST as a contrastive system for HiPDT, but we do not present experimental results with hypergraph representations. Hypergraphs are widely used by the SMT community, and discussions and contrastive experiments between HiFST and cube pruning decoders are available in the literature (Iglesias et al. 2009a; de Gispert et al. 2010).

We will propose a two-pass translation decoding strategy for HiPDT based on entropy-pruned first-pass language models.

Our complexity analysis prompts us to investigate decoding strategies based on large translation grammars and small language models. We describe, implement and evaluate a two-pass decoding strategy for a large-scale translation task using HiPDT (Section 5). We show that entropy-pruned language models can be used in first-pass translation, followed by admissible beam pruning of the output lattice and subsequent rescoring with a full language model. We analyze the search errors that might be introduced by a two-pass translation approach and show that these can be negligible if pruning thresholds are set appropriately (Section 5.2). Finally, we detail the experimental conditions and speed/performance tradeoffs that allow HiPDT to achieve state-of-the-art performance for large-scale SMT under a large grammar (Section 5.3), including lattice rescoring steps under a vast 5-gram language model and LMBR decoding (Section 5.4).

With this translation strategy HiPDT can yield very good translation performance. For comparison, the performance of this Chinese-to-English SMT described in Section 5.4 is equivalent to that of the University of Cambridge submission to the NIST OpenMT 2012 Evaluation[1].

2. Pushdown Automata

Informally, pushdown transducers are finite-state transducers that have been augmented with a stack. Typically 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 addition to the usual input and output labels (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/output 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 it 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. In this section we formally define pushdown automata and transducers.

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, see (Berstel 1979) for a more detailed presentation.

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 follows. $c_A(x)$ is the string obtained by

---
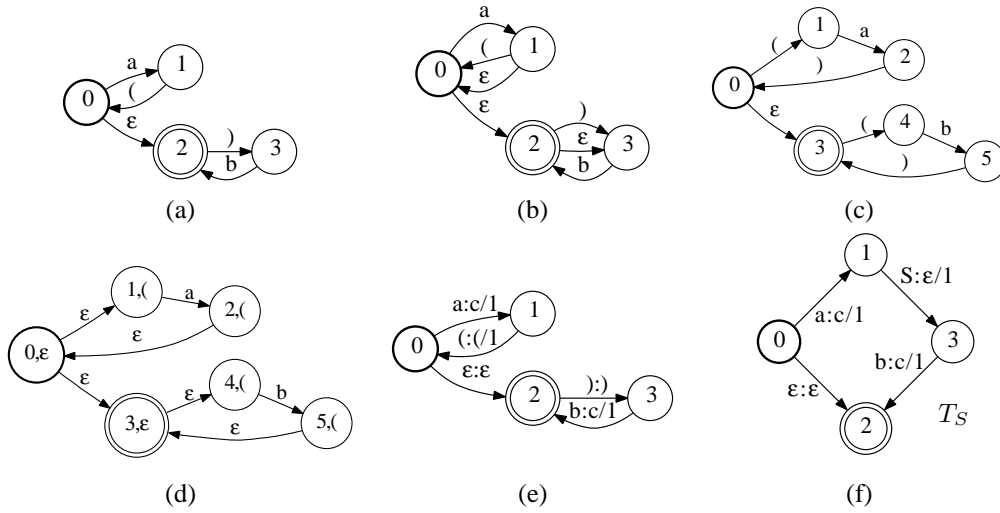
Figure 5
PDA Examples: (a) Non-regular PDA accepting $\{a^n b^n | n \in \mathbb{N}\}$. (b) Regular (but not bounded-stack) PDA accepting $a^* b^*$. (c) Bounded-stack PDA accepting $a^* b^*$ and (d) its expansion as an FSA. (e) Weighted PDT T_1 over the tropical semiring representing the weighted transduction $(a^n b^n, c^{2n}) \mapsto 3n$ and (f) equivalent RTN $(\{S\}, \{a, b\}, \{c\}, \{T_S\}, S)$.

iteratively deleting from x all factors of the form $a\bar{a}$ with $a \in A$. Observe that $D_A = c_A^{-1}(\epsilon)$. Finally, for a subset $B \subseteq A$, 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 semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a ring that may lack negation. It is specified by a set of values $\mathbb{K}$, two binary operations $\oplus$ and $\otimes$, and two designated values $\overline{0}$ and $\overline{1}$. The operation $\oplus$ is associative, commutative, and has $\overline{0}$ as identity. The operation $\otimes$ is associative, has identity $\overline{1}$, distributes with respect to $\oplus$, and has $\overline{0}$ as annihilator: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. If $\otimes$ is also commutative, we say that the semiring is *commutative*.

The *probability semiring* $(\mathbb{R}_+, +, \times, 0, 1)$ is used when the weights represent probabilities. The *log semiring* $(\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$, isomorphic to the probability semiring via the negative-log mapping, is often used in practice for numerical stability. The *tropical semiring* $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ is derived from the log semiring using the *Viterbi approximation*. These three semirings are commutative.

A *weighted pushdown automaton* (PDA) T over a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a 8-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, $E \subseteq Q \times (\Sigma \cup \widehat{\Pi} \cup \{\epsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions, and $\rho : F \to \mathbb{K}$ the final weight function. Let $e = (p[e], i[e], w[e], n[e])$ denote a transition in E; for simplicity, $(p[e], i[e], n[e])$ denotes an unweighted transition, i.e. a transition with weight $\overline{1}$.

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

$$T(x) = \bigoplus_{\pi \in P(x)} w[\pi] \otimes \rho(n[\pi]) \tag{1}$$

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 path π from I such that $c_{\Pi}(r_{\widehat{\Pi}}(i[\pi])) \in \Pi^*$:

$$|c_{\Pi}(r_{\widehat{\Pi}}(i[\pi]))| \leq K. \tag{2}$$

In other words, the number of open parentheses that are not closed along π is bounded. If T has a bounded stack, then it represents a regular language. Figure 5 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.

Finally, a *weighted pushdown transducer* (PDT) T over a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a 9-tuple $(\Sigma, \Delta, \Pi, \overline{\Pi}, Q, E, I, F, \rho)$ where Σ is the finite input alphabet, Δ is the finite output 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, $E \subseteq Q \times (\Sigma \cup \widehat{\Pi} \cup \{\epsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions, and $\rho : F \to \mathbb{K}$ the final weight function. Let $e = (p[e], i[e], o[e], w[e], n[e])$ denote a transition in E. Note that a PDA can be seen as a particular case of a PDT where $i[e] = o[e]$ for all its transitions. For simplicity, our following presentation focuses on acceptors, rather than the more general case of transducers. This is adequate for the translation applications we describe, with the exception of the treatment of alignment in Section 4.3, for which the intersection algorithm for PDTs and FSTs is given in Appendix 1.
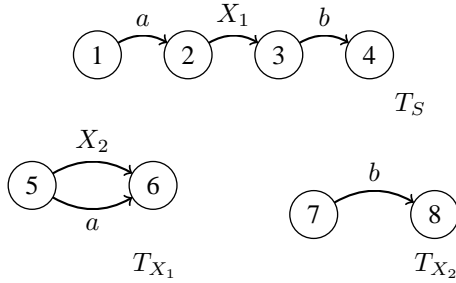
3. PDT Operations

In this section we describe in detail the following PDA algorithms: *Replacement*, *Composition*, *Shortest Path* and *(Pruned) Expansion* . While these are needed to implement HiPDT, these are general purpose algorithms, and suitable for many other applications outside the focus of this paper. 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). In this section, in order to simplify the presentation we will only consider machines over the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$. However, for each operation, we will specify in which semirings it can be applied.
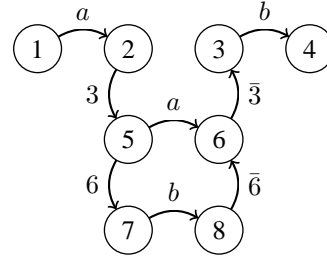
3.1 Recursive Transition Networks

We briefly give formal definitions for RTNs which will be needed to present the RTN expansion operation. Examples are shown earlier in Figures 1(b) and 3(a). Informally, an RTN is an automaton where some labels, nonterminals, are recursively replaced by other automata. We give the formal definition for acceptors; the extension to RTN transducers is straightforward.

An RTN R over the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ is a 4-tuple $(N, \Sigma, (T_\nu)_{\nu \in N}, S)$ where N is the alphabet of nonterminals, Σ is the input alphabet, $(T_\nu)_{\nu \in N}$ is a family of FSTs with input alphabet $\Sigma \cup N$, and $S \in N$ is the root nonterminal.

RTN R PDT T

R accepts $a\,a\,b$ and $a\,b\,b$. T accepts $a\,3\,a\,\bar{3}\,b$ and $a\,3\,6\,b\,\bar{6}\,\bar{3}\,b$.

Figure 6
Conversion of an RTN R to a PDA T by the replacement operation of Section 3.2. Using the notation of Section 2.1, in this example $\Pi = \{3, 5\}$ and $\overline{\Pi} = \{\bar{3}, \bar{5}\}$, with $f(3) = \bar{3}$ and $f(5) = \bar{5}$. The unweighted transition $(2, X_1, 3)$ in R is deleted and replaced by two new transitions $(2, 3, 5)$ and $(6, \bar{3}, 3)$; similarly $(5, X_2, 6)$ is replaced by $(5, 6, 7)$ and $(8, \bar{6}, 6)$. After application of the r_Σ mapping, the strings accepted by R and by T are the same.

A sequence $x \in \Sigma^*$ is accepted by (R, ν) if there exists an accepting path π in T_ν such that $\pi = \pi_1 e_1 \ldots \pi_n e_n \pi_{n+1}$ with $i[\pi_k] \in \Sigma^*$, $i[e_k] \in N$ and such that there exists sequences x_k such that x_k is accepted by $(R, i[e_k])$ and $x = i[\pi_1]x_1 \ldots i[\pi_n]x_n i[\pi_{n+1}]$. We say that x is accepted by R when it is accepted by (R, S). The weight associated by (R, ν) (and by R) to x can be defined in the same recursive manner.

As an example of testing whether an RTN accepts a sequence, consider the RTN R of Figure 6 and the sequence $x = a\,a\,b$. The path in the automata T_S can be written as $\pi = \pi_1\,e_1\,\pi_2$, with $i[\pi_1] = a$, $i[e_1] = X_1$, and $i[\pi_2] = b$. In addition, the machine $(R, i[e_1])$ accepts $x_1 = a$. Since $x = i[\pi_1]\,x_1\,i[\pi_2]$, it follows that x is accepted by (R, S).

3.2 Replacement

This algorithm converts a recursive transition network (RTN) into a PDA. As explained in Section 1.1, this PDT operation is applied by the HiPDT decoder in Step 1, and examples are given in earlier sections, e.g. in Figures 1 and 3.

Replacement acts on every transition of the RTN that is associated with a nonterminal. The source and destination states of these transitions are used to define the matched opening and closing parentheses, respectively, in the new PDA. Each RTN nonterminal transition is deleted and replaced by two new transitions which lead to and from the automaton indicated by the nonterminal. These new transitions have matched parentheses, taken from the source and destination states of the RTN transition they replace. Figure 6 gives a simple example.

Formally, given an RTN R, defined as $(N, \Sigma, (T_\nu)_{\nu \in N}, S)$, its replacement is the PDA T equivalent to R defined by the 8-tuple $(\Sigma, \Pi, \overline{\Pi}, Q, E, I, F, \rho)$ with $Q = \Pi = \bigcup_{\nu \in N} Q_\nu$, $I = I_S$, $F = F_S$, $\rho = \rho_S$, and $E = \bigcup_{\nu \in N} \bigcup_{e \in E_\nu} E^e$ where $E^e = \{e\}$ if $i[e] \notin N$ and

$$E^e = \{(p[e], n[e], \epsilon, w[e], I_\mu), (f, \overline{n[e]}, \epsilon, \rho_\mu(f), n[e]) | f \in F_\mu\} \tag{3}$$

with $\mu = i[e] \in N$ otherwise.
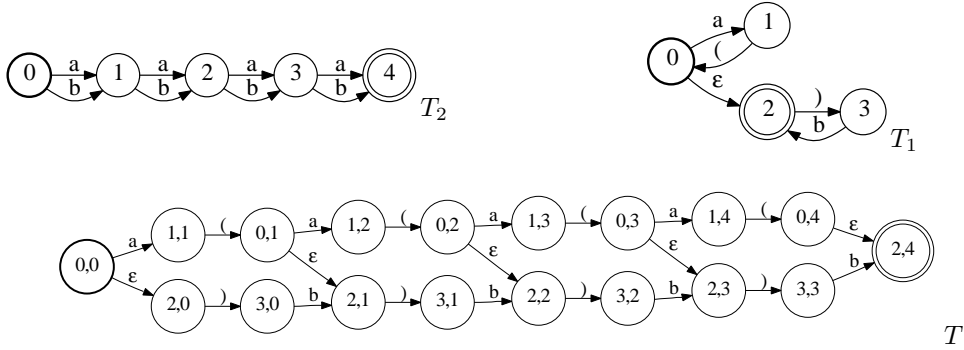
Figure 7
Composition example: composition of a PDA T_1 accepting $\{a^n, b^n\}$ with an FSA T_2 accepting $\{a, b\}^4$ to produce a PDA $T = T_1 \cap T_2$. T has only one balanced path, and this path accepts $a(a(\epsilon)b)b$.
Composition is performed by the PDA-FSA intersection described in Section 3.3.

The complexity of the construction is in $O(|T|)$. If $|F_\nu| = 1$ for all $\nu \in N$, 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. In this paper, we assume this optimization is always performed. We note here that RTNs can be defined and the replacement operation can be applied in any semiring.

3.3 Composition

Once we have created the PDA with translation scores, Step 2 in Section 1.1 applies the language model scores to the translation space. This is done by composition with an FSA containing the relevant language model weights.

The class of weighted pushdown transducers is closed under composition with weighted finite-state transducers (Bar-Hillel, Perles, and Shamir 1964; Nederhof and Satta 2003). Open-FST supports composition between automata T_1 and T_2, where T_1 is a weighted pushdown transducer and T_2 is a weighted finite-state transducer. If both T_1 and T_2 are acceptors, rather than transducers, the composition of a PDA and an FSA produces a PDA containing their intersection, and so no separate intersection algorithm is required for these automata. Given this, we describe only the simpler, special case of intersection between a PDA and an FSA, as this is sufficient for most of the translation applications described in this paper. The alignment experiments of Section 4.3 do require composition of transducers; the algorithm for composition of transducers is given in Appendix 1.

An example of composition by intersection is given in Figure 7. The states of T are created as the product of all the states in T_1 and T_2. Transitions are added as illustrated in Figure 8. These correspond to all paths through T_1 and T_2 that can be taken by a synchronized reading of strings from $\{a, b\}^*$. The algorithm is very similar to the composition algorithm for finite-state transducers, the difference being the handling of the parentheses. The parenthesis-labeled transitions are treated similarly to epsilon transitions, but the parenthesis labels are preserved in the result. This adds many unbalanced paths to T. In this example, T has five paths but only one balanced path, so that T accepts the string $a\,a\,b\,b$.

Formally, given a PDA $T_1 = (\Sigma, \Pi, \overline{\overline{\Pi}}, Q_1, E_1, I_1, F_1, \rho_1)$ and an FSA $T_2 = (\Sigma, Q_2, E_2, I_2, F_2, \rho_2)$, intersection constructs a new PDA $T = (\Sigma, \Pi, \overline{\overline{\Pi}}, Q, E, I, F, \rho)$, where $T = T_1 \cap T_2$ as follows:

PDT T_1	FSA T_2	PDT $T = T_1 \cap T_2$	Input Symbols
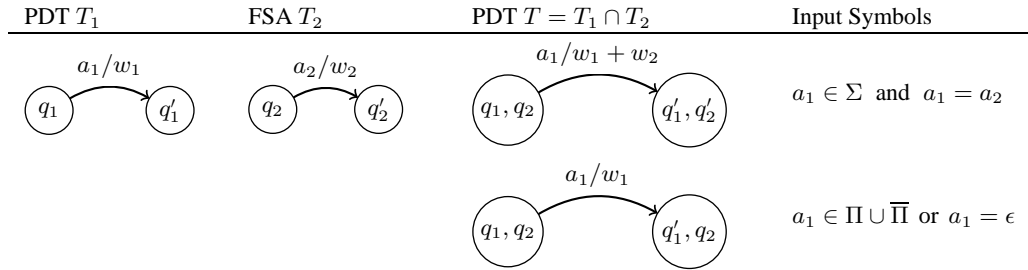
Transitions are added to T if and only if the conditions on the input symbols are satisfied.

Figure 8
PDA-FSA intersection under the tropical semiring. The PDA T is created by the intersection of the PDA T_1 and the FSA T_2, i.e. $T = T_1 \cap T_2$.

1. The new state space is in the product of the input state spaces: $Q \subset Q_1 \times Q_2$
2. The new initial and final states are $I = (I_1, I_2)$, and $F = \{(q_1, q_2) : q_1 \in F_1, q_2 \in F_2\}$
3. Weights are assigned to final states $(q_1, q_2) \in Q$ as $\rho(q_1, q_2) = \rho(q_1) + \rho(q_2)$
4. For pairs of transitions $(q_1, a_1, w_1, q_1') \in E_1$ and $(q_2, a_2, w_2, q_2') \in E_2$, a transition is added between states (q_1, q_2) and (q_1', q_2') as specified in Figure 8.

The intersection algorithm given here assumes that T_2 has no input-ϵ transitions. When T_2 has input-ϵ transitions, an epsilon filter (Mohri 2009; Allauzen, Riley, and Schalkwyk 2011) generalized to handle parentheses can be used. Note that steps 1 and 2 do not require the construction of all possible pairs of states; only those states reachable from the initial state and needed in step 4 are actually generated. The complexity of the algorithm is in $O(|T_1|\,|T_2|)$ in the worst case, as will be discussed in Section 4. Composition requires the semiring to be commutative.

3.4 Shortest Distance and Path Algorithms

With a PDA including both translation and language model weights, HiPDT can extract the best translation (Step 3a in Section 1.1). To this end, a general PDA shortest distance/path algorithm is needed.

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. Figure 10 gives a pseudo-code description of the shortest-distance algorithm, which we now discuss.

Given a PDA $T = (\Sigma, \Pi, \overline{\Pi}, Q, E, I, F, \rho)$, the GETDISTANCE($T$) algorithm computes the shortest distance from the start state I to the final state[2] $f \in F$. The algorithm recursively calculates

$d[q, q'] \in \mathbb{K}$ – the shortest distance from state q to state q' along a balanced path

At termination, the algorithm returns $d[I, f]$ as the cost of the shortest path through T.

---

2 For simplicity, we assume T has only one final state.

The core of the shortest distance algorithm is the procedure GETDISTANCE(T, s) which calculates the distances $d[s, q]$ for all states q that can be reached from s. For an FSA, this procedure is called once, as GETDISTANCE(T, I), to calculate $d[I, q] \, \forall q$.

For a PDA, the situation is more complicated. 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 by a balanced path starting from s. If s has several incoming open parenthesis transitions, a naive implementation might lead to the states in C_s to be visited exponentially many times. This is avoided by memoizing the shortest distance from s to states in C_s. To do this, GETDISTANCE(T, s) calculates $d[s, s']$ for all $s' \in C_s$, and it also constructs sets of transitions

$$B[s, a] = \{e \in E : p[e] \in C_s \text{ and } i[e] = \overline{a}\} \quad \forall a \in \Pi \tag{4}$$

These are the transitions with label $\overline{a}$ leaving states in C_s.

Consider any incoming transition to s, (q, a, w, s), with $a \in \Pi$. For every transition $e' = (s', \overline{a}, w', q')$, $e' \in B[s, a]$ the following holds [3]

$$d[q, q'] = w + d[s, s'] + w' \tag{5}$$

If $d[s, s']$ is available, the shortest distance from q to q' along any balanced path through s can be computed trivially by Equation 5. For any state s with incoming open parenthesis transitions, only a single call to GETDISTANCE(T, s) is needed to precompute the necessary values.

Figure 9 gives an example. When transition $(2, (_1, 0, 5)$ is processed, GETDISTANCE$(T, 5)$ is called. The distance $d[5, 7]$ is computed, and following transitions are logged: $B[5, (_1] \leftarrow \{(7,)_1, 0, 8)\}$ and $B[5, (_2] \leftarrow \{(7,)_2, 0, 9)\}$. Later, when the transition $(4, (_2, 0, 5)$ is processed, its matching transition $(7,)_2, 0, 9)$ is extracted from $B[5, (_2]$. The distance $d[4, 9]$ is then found by Equation 5 as $d[5, 7]$. This avoids redundant re-calculation of distances along the shortest balanced path from state 4 to state 9.

We now briefly discuss the shortest distance pseudo-code given in Figure 10. The description may be easier to follow after reading the worked example in Figure 9. Note that the sets C_s are not computed explicitly by the algorithm.

The shortest distance calculation proceeds as follows. Self-distances, i.e. $d[q, q]$, are set initially to ∞; when GETDISTANCE(T, q) is called it sets $d[q, q] = 0$ to note that q has been visited. 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 (lines 7-11). Transitions labeled by non-parenthesis are treated as in Mohri (2009) (line 7-8). When a transition e is labeled by a close parenthesis, e is added to $B[s, \overline{i[e]}]$ to indicate that this transition balances all incoming open parentheses into s labeled by $\overline{i[e]}$ (line 9-10). Finally, if e has an open parenthesis, and if its destination has not already been visited, a new instance of GETDISTANCE is started from $n[e]$ (line 12-13). The destination states of all transitions balancing e are then relaxed (lines 14-16).

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 accumulated number of transitions examined at line 16 is in $O(Z|Q| \, |E|^2)$ in the worst case, where Z 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

---

3 This assumes all paths from q to q' pass through s. The RELAX operation (Figure 10) handles the general case.
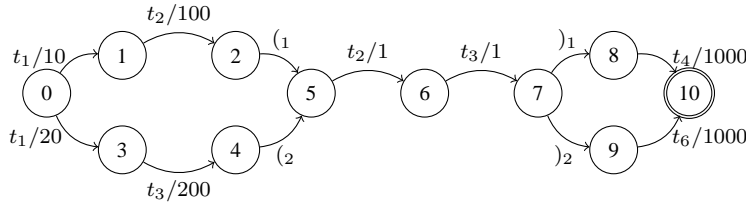
GETDISTANCE(T) runs
1. Initialisation: $d[q, q] \leftarrow \infty, \ \forall q \in Q$
2. GETDISTANCE($T, 0$) is called
   GETDISTANCE($T, 0$) runs
   3. Distances are calculated from state 0:
      $d[0, 0] \leftarrow 0; d[0, 1] \leftarrow d[0, 0] + w[0, 1]; d[0, 2] \leftarrow d[0, 1] + w[1, 2]$
   4. Transition $e_1 = (2, (_1, 0, 5)$ is reached. e_1 has symbol $i[e_1] = (_1$ and destination state $n[e_1] = 5$
   5. $d[5, 5] = \infty$ so GETDISTANCE($T, 5$) is called
      GETDISTANCE($T, 5$) runs
      6. Distances are calculated from state 5:
         $d[5, 5] \leftarrow 0; d[5, 6] \leftarrow d[5, 5] + w[5, 6]; d[5, 7] \leftarrow d[5, 6] + w[6, 7]$
      7. The transitions $(7,)_1, 0, 8)$ and $(7,)_2, 0, 9)$ are reached and memoized
         $B[5, (_1] \leftarrow \{(7,)_1, 0, 8)\}$
         $B[5, (_2] \leftarrow \{(7,)_2, 0, 9)\}$
      GETDISTANCE($T, 5$) ends
   GETDISTANCE($T, 0$) resumes.
   8. Transition $e_1 = (2, (_1, 0, 5)$ is still being processed, with $p[e_1] = 2$, $n[e_1] = 5$, and $i[e_1] = (_1$
   9. Transition $e_2 = (7,)_1, 0, 8)$ matching $(_1$ is extracted from $B[n[e_1], i[e_1]]$, with $p[e_2] = 7$ and $n[e_2] = 8$
   10. Distance $d[0, 8]$ is calculated as $d[0, n[e_2]]$:
      $d[0, n[e_2]] \leftarrow d[0, p[e_1]] + w[p[e_1], n[e_1]] + d[n[e_1], p[e_2]] + w[p[e_2], n[e_2]]$
   10. Processing of e_1 finishes, and calculation of distances from 0 continues:
      $d[0, 10] \leftarrow d[0, 8] + w[8, 10]$
      10 is a final state. Processing continues with transition $(0, t_1, 20, 3)$
      $d[0, 3] \leftarrow d[0, 0] + w[0, 3]; d[0, 4] \leftarrow d[0, 3] + w[3, 4]$
   13. Transition $e_3 = (4, (_2, 0, 5)$ is reached.
      e_3 has symbol $i[e_3] = (_2$, source state $p[e_3] = 4$, and destination state $n[e_3] = 5$
   14. GETDISTANCE($T, 5$) is not called, since $d[5, 5] = 0$ indicates state 5 has been previously visited
   15. Transition $e_4 = (7,)_2, 0, 9)$ matching $(_2$ is extracted from $B[n[e_3], i[e_3]]$, with $p[e_4] = 7$ and $n[e_4] = 9$
   16. Distance $d[0, 9]$ is calculated as $d[0, n[e_4]]$, using cached values:
      $d[0, n[e_4]] \leftarrow d[0, p[e_3]] + w[p[e_3], n[e_3]] + d[n[e_3], p[e_4]] + w[p[e_4], n[e_4]]$
   17. $d[0, 10]$ is less than ∞ :
      $d[0, 10] \leftarrow \min(d[0, 10], d[0, 9] + w[9, 10])$
18. GETDISTANCE($T, 0$) ends and returns $d[0, 10]$
GETDISTANCE(T) ends

Figure 9
Step-by-step description of the shortest distance calculation for the given PDA by the algorithm of
Figure 10. For simplicity, $w[q, q']$ indicates the weight of the transition connecting q and q'.

be expressed as $O(Z|T|^3 \, \Gamma(|T|))$. 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.

As was shown in Section 3.2, when T has been obtained by converting an RTN or a
hypergraph into a PDA, 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
$d[s, q]$ is non-infinity. Moreover, for each open parenthesis transition e, there exists a unique close

SHORTESTDISTANCE(T)
 1   **for** each $q \in Q$ and $a \in \Pi$ **do**
 2       $B[q, a] \leftarrow \emptyset$
 3   **for** each $q \in Q$ **do**
 4       $d[q, q] \leftarrow \infty$
 5   GETDISTANCE(T, I) $\triangleright$ *I is the unique initial state*
 6   **return** $d[I, f]$ $\triangleright$ *f is the unique final state*

RELAX($s, q, w, \mathcal{S}$)
 1   **if** $d[s, q] > w$ **then** $\triangleright$ *if w is a better estimate of the distance from s to q*
 2       $d[s, q] \leftarrow w$ $\triangleright$ *update $d[s, q]$*
 3   **if** $q \notin \mathcal{S}$ **then** $\triangleright$ *enqueue q in $\mathcal{S}$ if needed*
 4   ENQUEUE($\mathcal{S}, q$)

GETDISTANCE(T, s)
 1   **for** each $q \in Q$ **do**
 2       $d[s, q] \leftarrow \infty$
 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** $\triangleright$ *$E(q)$ is the set of transitions leaving state q*
 9   **if** $i[e] \in \Sigma \cup \{\epsilon\}$ **then** $\triangleright$ *$i[e]$ is a regular symbol*
10   RELAX($s, n[e], d[s, q] + w[e], \mathcal{S}_s$)
11   **elseif** $i[e] \in \overline{\Pi}$ **then** $\triangleright$ *$i[e]$ is a close parenthesis*
12               $B[s, \overline{i[e]}] \leftarrow B[s, \overline{i[e]}] \cup \{e\}$
13   **elseif** $i[e] \in \Pi$ **then** $\triangleright$ *$i[e]$ is an open parenthesis*
14   **if** $d[n[e], n[e]] = \infty$ **then** $\triangleright$ *$n[e]$ is the destination state of transition e*
15   GETDISTANCE($T, n[e]$)
16   **for** each $e' \in B[n[e], i[e]]$ **do**
17                   $w \leftarrow d[s, q] + w[e] + d[n[e], p[e']] + w[e']$
18   RELAX($s, n[e'], w, \mathcal{S}_s$)

Figure 10
PDT shortest distance algorithm.

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 $O(|T|)$ in time and space.

The algorithm can be modified (without changing the complexity) to compute the shortest path by keeping track of parent pointers. The notion of shortest path requires the semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ to have the *path property*: for all a, b in $\mathbb{K}$, $a \oplus b \in \{a, b\}$. The shortest-distance operation as presented here and the shortest-path operation can be applied in any semiring having the path property by using the natural order defined by $\oplus$: $a \leq b$ iff $a \oplus b = a$. However, the shortest distance algorithm given in Figure 10 can be extended to work for k-closed semirings using the same techniques that were used by Mohri (2002).

The shortest distance in the intersection of a string s and a PDA T determines if T recognizes s. PDA recognition is closely related to CFG parsing; a CFG can be represented as a PDT whose input recognizes the CFG and whose output identifies the parse (Aho and Ullman 1972). Lang (Lang 1974) showed that the cubic tabular method of Earley can be naturally applied to PDAs; others give the weighted generalizations (Stolcke 1995; Nederhof and Satta 2006). Earley's algorithm has its analogs in the algorithm in Figure 10: the *scan* step corresponds to taking a non-parenthesis transition at line 10, the *predict* step to taking an open parenthesis at lines 14-15 and the *complete* step to taking the closed parentheses at lines 16-18.

Specialization to Translation. Following the formalism of Section 1, we are interested in applying shortest distance and shortest path algorithms to automata created as $L = T_p \cap M$, where T_p, the translation representation, is a PDA derived from an RTN (via replacement) and M, the language model, is a finite automaton.

For this particular case, the time complexity is $O(|T_p||M|^3)$ and the space complexity is $O(|T_p||M^2|)$. The dependence on $|T_p|$ is linear, rather than cubic or quadratic. The reasoning is as follows. 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.

3.5 Expansion

As explained in Section 1.1, HiPDT can apply Step 3b to generate translation lattices. This step is typically required for any posterior lattice rescoring strategies. We first describe the unpruned expansion. However, in practice a pruning strategy of some sort is required to avoid state explosion. Therefore, we also describe an implementation of the PDA expansion that includes admissible pruning under a likelihood beam, thus controlling on-the-fly the size of the output lattice.

3.5.1 Full Expansion. Given a bounded-stack PDA T, the expansion of T is the FSA T' equivalent to T. A simple example is given in Figure 11.

Expansion starts from the PDA initial state. States and transitions are added to the FSA as the expansion proceeds along paths through the PDA. In the new FSA, parentheses are replaced by epsilons, and as open parentheses are encountered on PDA transitions, they are 'pushed' into the FSA state labels; in this way the stack depth is maintained along different paths through the PDA. Conversely, when a closing parenthesis is encountered on a PDA path, a corresponding opening parenthesis is 'popped' from the FSA state label; if this is not possible, e.g. as in state $(5, \epsilon)$ in Figure 11, expansion along that path halts.

The resulting automata accept the same language. The FSA topology changes, typically with more states and transitions than the original PDA, and the number of added states is controlled only by the maximum stack depth of the PDA.

Formally, suppose the PDA $T = (\Sigma, \Pi, \overline{\Pi}, Q, E, I, F, \rho)$ has a maximum stack depth of K. The set of states in its FSA expansion T' are then

$$Q' = \{(q, z) : q \in Q, \, z \in \Pi^* \text{ and } |z| \leq K\} \tag{6}$$

and T' has initial state (I, ϵ) and final states $F' = \{(q, \epsilon) : q \in F\}$. The condition that T has a bounded stack ensures that Q' is finite. Transitions are added to T' as described in Figure 12.

The full expansion operation can be applied to PDA over any semiring. The complexity of the algorithm is linear in the size of T'. However, the size of T' can be exponential in the size of T, which motivates the development of pruned expansion as discussed next.

3.5.2 Pruned Expansion. Given a bounded-stack PDA T, the *pruned expansion* of T with threshold β is an FST T'_β obtained by deleting from T' all states and transitions that do not belong to any accepting path π in T' such that $w[\pi] \otimes \rho[\pi] \leq d + \beta$, where d is the shortest distance in T.

A naive implementation consisting of fully expanding T and then applying the FST pruning algorithm would lead to a complexity in $O(|T'| \log |T'|) = O(e^{|T|}|T|)$. Assuming that the reverse T^R of T is also bounded-stack, an algorithm whose complexity is in $O(|T| \, |T'_\beta| + |T|^3 \log |T|)$
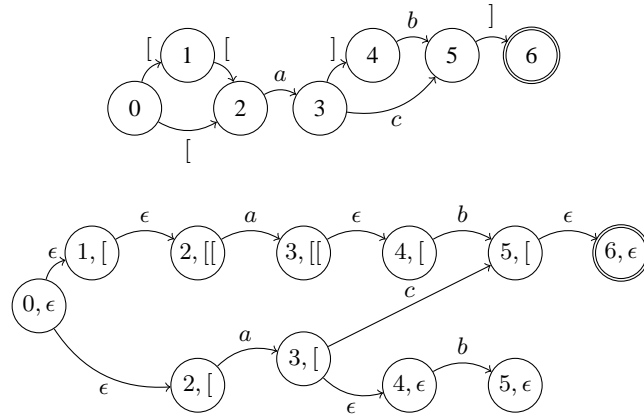
Figure 11
Full expansion of a PDA to an equivalent FSA. The PDA maximum stack depth is 2, therefore the FSA states belong to $\{0, .., 6\} \times \{\epsilon, [, [[\}$. Expansion can create incomplete paths in the FSA, e.g. corresponding here to the unbalanced PDA path $[\,a\,]\,b\,]$, however these are guaranteed to be unconnected, i.e. not to lead to a final state. Any unconnected states are removed after expansion.

| Transition in PDA T	New transition in FSA T'	Conditions	Explanation
	$q, z \xrightarrow{a/w} q', z$	$a \in \Sigma \cup \{\epsilon\}$	a is not a parenthesis; stack depth is unchanged
$q \xrightarrow{a/w} q'$	$q, z \xrightarrow{\epsilon} q', za$	$a \in \Pi$	a is an open parenthesis; an epsilon transition is added, and a is 'pushed' into the destination state, increasing the stack depth
	$q, z'\overline{a} \xrightarrow{\epsilon} q', z'$	$a \in \overline{\Pi}$	a is an closing parenthesis; an epsilon transition is added, and the matching open parenthesis $\overline{a}$ is 'popped' from the destination state, decreasing the stack depth

Figure 12
PDA Expansion. A states (q, z) and (q', z') in the FSA T' will be connected by an transition if and only if the above conditions hold on the corresponding transition between q and q' in the PDA T.

can be obtained by first applying the shortest distance algorithm from the previous section to T^R and then using this to prune the expansion as it is generated. To simplify the presentation, we assume that $F = \{f\}$ and $\rho(f) = 0$.

The motivation for using reversed automaton in pruning is easily seen by looking at FSAs. For an FSA, the cost of the shortest path through an transition (q, x, w, q') can be stated as
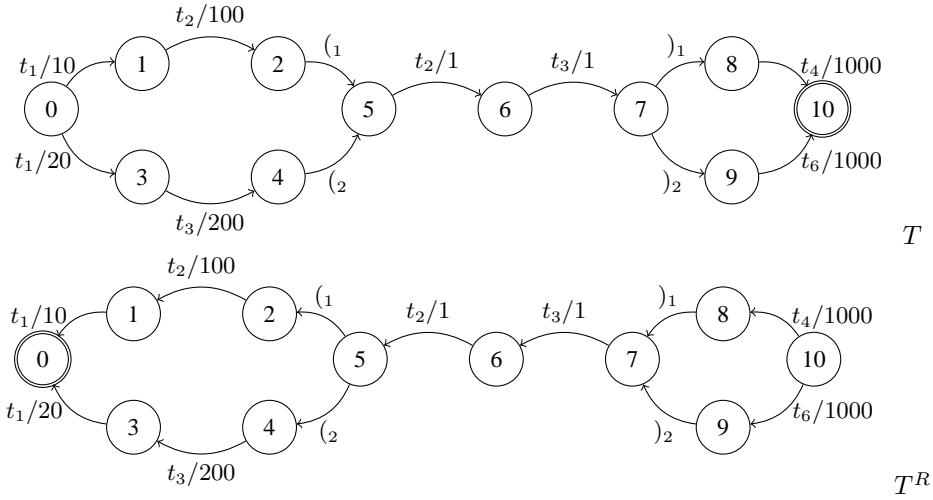
Figure 13
PDA T and its reverse T^R. T^R has start state 10, final state 0, $\Pi^R = \{)_1,)_2\}$, and $\overline{\Pi}^R = \{(_1, (_2\}$.

$d[I, q] + w + d[q', f]$. Distances $d[I, q]$, i.e. distances from the start state, are computed by the shortest distance algorithm, as discussed in Section 3.4. However distances of the form $d[q', f]$ are not readily available. To compute these, a shortest distance algorithm is run over the reversed automaton. Reversal preserves states and transitions, but swaps the source and destination state (see Figure 13 for a PDA example). The start state in the reversed machine is f, so that distances are computed from f; these are denoted $d^R[f, q]$ and correspond to $d[q, f]$ in the original FSA. The cost of the shortest path through an FSA transition (q, x, w, q') can then be computed as $d[I, q] + w + d^R[f, q']$.

Calculation for PDAs is more complex. Transitions with parentheses must be handled such that distances through them are calculated over balanced paths. For example, if T in Figure 13 was an FSA, the shortest cost of any path through the transition $e = (4, (_2, 0, 5)$ could be calculated as $d[0, 4] + 0 + d[5, 10]$. However this is not correct, because $d[5, 10]$, the shortest distance from 5 to 10, is found via a path through the transition $(7,)_1, 0, 8)$.

Correct calculation of the minimum cost of balanced paths through PDA transitions can be done using quantities computed by the PDA shortest distance algorithm. For a PDA transition $e = (q, a, w, q')$, $a \in \Pi$, the cost of the shortest balanced path through e can be found as [4]

$$c(e) = d[I, q] + w[e] + \min_{e' \in B[q', a]} d[q', p[e']] + w[e'] + d^R[n[e'], f] \tag{7}$$

where $B[q', a]$ and $d[p[e'], q']$ are computed by the PDA shortest distance algorithm over T, and $d^R[n[e'], f]$ is computed by the PDA shortest distance algorithm over T^R.

In Figure 13, the shortest cost of paths through the transition $e = (4, (_2, 0, 5)$ is found as follows: the shortest distance algorithm over T calculates $d[0, 4] = 220$, $d[5, 7] = 2$, and $B[5, (_2] = \{7,)_2, 0, 9\}$; the shortest distance algorithm over T^R calculates $d^R[10, 9] = 1000$

---

4 Note that $d[p[e'], q']$ could be replaced by $d^R[q', p[e']]$.

(trivially, here); the cost of the shortest path through e is

$$d[0,4] + w[e] + d[5,7] + w[e'] + d^R[10,9] = 220 + 0 + 2 + 0 + 1000$$

Pruned expansion is therefore able to avoid expanding transitions that would not contribute to any path that would survive pruning. Prior to expansion of a PDA T to an FSA T', the shortest distance d in T is calculated. Transitions $e = (q, a, w, q')$, $a \in \Pi$, are expanded as transitions $e = ((q, z), q, w, (q', za))$ in T' only if $c(e) \leq d + \beta$, as calculated by Equation 7.

The pruned expansion algorithm implemented in OpenFST is necessarily more complicated than the simple description given above. Pseudo-code describing the OpenFST implementation is given in Appendix 2.

The pruned expansion operation can be applied in any semiring having the path property.

4. HiPDT Analysis and Experiments: Computational Complexity

We now address the following questions:

- What are the differences between the FSA and PDA representations as observed in a translation/alignment task? How do their respective decoding algorithms perform in relation to the complexity analysis described above? How many times is exact decoding achievable in each case?

We will discuss the complexity of both HiPDT and HiFST decoders as well as the hypergraph representation, with an emphasis on Hiero-style SCFGs. We assess our analysis for FSA and PDA representations by contrasting HiFST and HiPDT with large grammars for translation and alignment. For convenience, we refer to the hypergraph representation as T_h, and to the FSA and PDA representations as T_f and T_p.

We first analyze 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 representations 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 RTN expansion to FSA 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, Perles, and Shamir 1964) with time and space complexity $O(|T_h||M|^{l+1})$ where l is the maximum number of symbols on the right-hand side of a grammar rule in T_h. Dyer (2010a) presents a more practical intersection algorithm that avoids creating rules that are inaccessible from the start symbol. With deterministic M, the intersection complexity becomes $O(|T_h||M|^{l_N+1})$ where l_N is the rank of the SCFG, i.e. l_N is the maximum number of nonterminals on the right-hand side of a grammar rule. With Hiero-styles rules, $l_N = 2$ so the complexity is $O(|T_h||M|^3)$ in that case.[5] The PDA intersection algorithm from Section 3.3 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

---

5 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)).

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
Translation complexity of target language representations for translation grammars of rank 2.

2009). As presented in Section 3.4, the PDA representation can require time cubic and space quadratic in $|M|$.

Table 1 summarizes the complexity results for SCFGs of rank 2. The PDA representation is equivalent in time and superior in space complexity to the CFG/hypergraph representation, in general, and it can be superior in both space and time to the FSA representation depending on the relative SCFG and LM sizes. The FSA representation favors smaller target translation grammars and larger language models.

In practice, the PDA and FSA representations benefit greatly from the optimizations mentioned previously (Figure 3 and accompanying discussion). For the FSA representation, these operations can offset the exponential dependencies in the worst-case complexity analysis. For example, in translation of a 15 word sentence taken at random from the development sets described later, expansion of an RTN yields a WFSA with 174×10^6 states. By contrast, if the RTN is determinized and minimized prior to expansion, the resulting WFSA has only 34×10^3 states. Size reductions of this magnitude are typical. In general, the original RTN, hypergraph or CFG representation can be exponentially larger than the RTN/PDT optimized as described.

Although our interest is primarily in Hiero-style translation grammars, which have rank 2 and a relatively small number of nonterminals, this complexity analysis can be extended to other grammars. For SCFGs of arbitrary rank l_N, translation complexity in time for hypergraphs becomes $O(|G||s|^{l_N+1}|M|^{l_N+1})$; with FSAs the time complexity becomes $O(e^{|G||s|^{l_N+1}}|M|)$; and with PDAs the time complexity becomes $O(|G||s|^{l_N+1}|M|^3)$. For more complex SCFGs with rules of rank greater than 2, such as SAMT (Zollmann and Venugopal 2006) or GHKM (Galley et al. 2004), this suggests that PDA representations may offer computational advantages in the worst case relative to hypergraph representations, although this must be balanced against other available strategies such as binarization (Zhang et al. 2006; Xiao et al. 2009) or scope pruning (Hopkins and Langmead 2010). Of course, practical translation systems introduce various pruning procedures to achieve much better decoding efficiency than the worst cases given here.

We will next describe the translation grammar and language model for our experiments, which will be used throughout the remaining of this paper (except when stated otherwise). In the following sections we assess the complexity discussion with a contrast between HiFST (FSA representation) and HiPDT (PDA representation) under large grammars.

4.1 Translation Grammars and Language Models

Translation grammars are extracted from a subset of the GALE 2008 evaluation parallel text[6]; 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 the

---

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

NIST evaluation sets MT02 through MT06 [7]. In tuning the systems, MERT (Och 2003) iterative parameter estimation under IBM BLEU[8] 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 published procedures (Chiang 2007; Iglesias et al. 2009b) to extract hierarchical phrases from the union of the directional word alignments. We call a translation grammar (G) the set of rules extracted from this process. For reference, the number of rules in G that can apply to the *tune-nw* is 1.1M, of which 593K are standard non-hierarchical phrases and 511K are strictly hierarchical rules.

We will use two English language models in these translation experiments. The first language model, denoted M_1, is a 4-gram estimated over 1.3B words taken from the target side of the parallel text and the AFP and Xinhua portions of the English Gigaword Fourth Edition (LDC2009T13). We use both Kneser-Ney (Kneser and Ney 1995) and Katz (Katz 1987) smoothing in estimating M_1. Where language model reduction is required, we apply Stolcke entropy pruning (Stolcke 1998) to M_1 under the relative perplexity threshold θ. The resulting language model is labeled as M_1^{θ}.

	θ	0	7.5×10^{-9}	7.5×10^{-8}	7.5×10^{-7}	7.5×10^{-6}	7.5×10^{-5}	7.5×10^{-4}	7.5×10^{-3}
KN	2-grams	28M	10M	2.5M	442K	37K	1.3K	21	0
	3-grams	61M	6M	969K	74K	2.7K	38	0	0
	4-grams	117M	3M	219K	5K	44	0	0	0
	perplexity	98.1	122.2	171.5	290.4	605.1	1270.2	1883.6	2200.0
KATZ	2-grams	28M	7M	2M	391K	52K	4K	117	1
	3-grams	64M	10M	1.5M	148K	8.4K	197	1	0
	4-grams	117M	4.6M	398K	19K	510	1	0	0
	perplexity	106.7	120.4	146.9	210.5	336.6	596.5	905.0	1046.1

Table 2
Number of n-grams with explicit conditional probability estimates assigned by the 4-gram language models M_1^{θ} after entropy pruning of M_1 at threshold values θ. Perplexities over the (concatenated) *tune-nw* reference translations are also reported. The Kneser-Ney and Katz 4-gram LM have 416190 unigrams, which are not removed by pruning.

The reduction in size in terms of component n-grams is summarized in Table 2. For aggressive enough pruning, the original 4-gram model can be effectively reduced to a trigram, bigram, or unigram models. For both the Katz and the Kneser-Ney four gram language models: at $\theta = 7.5E - 05$ the number of 4-grams in the LM is effectively reduced to zero; at $\theta = 7.5E - 4$ the number of 3-grams is effectively 0; and at $\theta = 7.5E - 3$, only unigrams remain. Development set perplexities increase as entropy pruning becomes more aggressive, with the Katz smoothed model performing better under pruning (Chelba et al. 2010; Roark, Allauzen, and Riley 2013).

We will also use a larger language model, denoted M_2, obtained by interpolating M_1 with a zero-cutoff stupid-backoff 5-gram model (Brants et al. 2007) estimated over 6.6B words of English newswire text; M_2 is estimated as needed for the n-grams required for the test sets.

Decoding with $G + M_1^\theta$ under a 10GB memory size limit						
#	θ	HiFST			HiPDT		
		Success	Failure		Success	Failure	
			Expansion	*Intersection*		*Intersection*	*Shortest Path*
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 3
Success in finding the 1-best translation under G with various M_1^θ under a memory size limit of 10GB as measured over *tune-nw* (1755 sentences). We note which operations in translation exceeded the memory limit: either Expansion and Intersection for HiFST, or Intersection and Shortest Path operation for HiPDT.

4.2 Exact Decoding with Large Grammars and Small Language Models

We now compare HiFST and HiPDT in translation with our large grammar G. In this case we know that exact search is often not feasible for HiFST.

We run both decoders over *tune-nw* with a restriction on memory use of 10 GB. If this limit is reached in decoding, the process is killed[9]. Table 3 shows the number of times each decoder succeeds in finding a hypothesis under the memory limit when decoding with various entropy-pruned LMs 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 difference in success rates between the decoders is more pronounced as the language model is more aggressively pruned: for $\theta = 7.5 \times 10^{-7}$ HiPDT succeeds for all but three sentences.

As Table 3 shows, HiFST fails most frequently in its initial expansion from RTN to FSA; this operation depends only on the translation grammar and does not benefit from any reduction in the language model size. Subsequent intersection of the FSA with the language model can still pose a challenge, although as the language model is reduced, this intersection fails less often. By contrast, HiPDT intersects the translation grammar with the language model prior to expansion and this operation nearly always finishes successfully. The subsequent shortest path (or pruned expansion) operation is prone to failure, but the risk of this can be greatly reduced by using smaller language models.

In the next section we contrast both HiPDT and HiFST for alignment.

4.3 Alignment with Inversion Transduction Grammars

We continue to explore applications characterized by large translation grammars G and small language models M. As an extreme instance of a problem involving a large translation grammar and a simple target language model, we consider parallel text alignment under an Inversion Transduction Grammar (ITG) (Wu 1997). This task, or something like it, is often done in translation grammar induction. The process should yield the set of derivations, with scores, which generate the target sentence as a translation of the source sentence. In alignment the target language model is extremely simple: it is simply an acceptor for the target language sentence so that $|M|$ is linear in the length of the target sentence. In contrast, the search space needs now to be represented with pushdown transducers (instead of pushdown automata) keeping track of

---

7 See http://www.itl.nist.gov/iad/mig/tests/mt/ .

8 See ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13.pl

9 We use the UNIX *ulimit* command. The experiment was carried out over machines with different configurations and loads and so these numbers should be considered as approximate values.

HiFST			HiPDT		
Success	Failure		Success	Failure	
	memory/time	grammar		memory/time	grammar
41%	53%	6%	89%	0%	11%

Table 4
Percentages of success and failure in aligning 2500 sentence pairs under G_{ITG} with HiFST and HiPDT.
HiPDT finds an alignment whenever it is possible under the translation grammar.

both translations and derivations, i.e. indices of the rules in the grammar (Iglesias et al. 2009a; de Gispert et al. 2010; Dyer 2010b).

We define a word-based translation grammar G_{ITG} for the alignment problem as follows. First, we obtain word-to-word translation rules of the form $X \rightarrow \langle s, t \rangle$ based on probabilities from IBM Model 1 translation tables estimated over the parallel text, where s and t are one source and one target word respectively ($\sim 16M$ rules). Then, we allow monotonic and inversion transduction of two adjacent nonterminals in the usual ITG style (ie. add $X \rightarrow \langle X_1\ X_2, X_1\ X_2 \rangle$ and $X \rightarrow \langle X_1\ X_2, X_2\ X_1 \rangle$). Additionally we allow unrestricted source word deletions ($X \rightarrow \langle s, \epsilon \rangle$), and restricted target word insertions ($X \rightarrow \langle X_1\ X_2, X_1\ t\ X_2 \rangle$). This restriction, which is solely motivated by efficiency reasons, disallows the insertion of two consecutive target words. We make no claims about the suitability or appropriateness of this specific grammar for either alignment or translation; we introduce this grammar only to define a challenging alignment task.

A set of 2500 sentence pairs of up to 50 source and 75 target words was chosen for alignment. These sentences come from the same Chinese-to-English parallel data described in Section 4.1. Hard limits on memory usage (10GB) and processing time (10 minutes) were imposed for processing each sentence pair. If HiPDT or HiFST exceeded either limit in aligning any sentence pair, alignment was stopped and a 'memory/time failure' was noted. Even if the resource limits are not exceeded, alignment may fail due to limitations in the grammar. This happens when either a particular word pair rule that is not in our Model 1 table, or more than one consecutive target insertions are needed to reach alignment. In such cases, we record a 'grammar failure', as opposed to a 'memory/time failure' failure.

Results are reported in Table 4. Of the 2500 sentence pairs, HiFST successfully aligns only 41% of the sentence pairs under these time and memory constraints. The reason for this low success rate is that HiFST must generate and expand all possible derivations under the ITG for a given sentence pair. Even if it is strictly enforced that the FSA in every CYK cell contains only partial derivations which produce substrings of the target sentence, expansion often exceeds the memory/time constraints. In contrast, HiPDT succeeds in aligning all sentence pairs that can be aligned under the grammar (89%), since it never fails due to memory or time constraints. In this experiment, if alignment is at all possible, HiPDT will find the best derivation. Alignment success rate (or coverage) could trivially be improved by modifying the ITG to allow more consecutive target insertions, or by increasing the number of word-to-word rules, but that would not change the conclusion in the contrast between HiFST and HiPDT.

The computational analysis from the beginning of this section applies to alignment. The language model M is replaced by an acceptor for the target sentence, and if we assume that the target sentence length is proportional to the source sentence length, it follows that $|M| \propto |s|$ and the worst-case complexity for HiPDT in alignment mode is $O(|s|^6|G|)$. This is comparable to ITG alignment (Wu 1997) and the intersection algorithm of Dyer (2010b).

Our experimental results support the complexity analysis summarized in Table 1. HiPDT is more efficient in ITG alignment and this is consistent with its linear dependence on the grammar size, whereas HiFST suffers from its exponential dependence. This use of PDAs in alignment

does not rely on properties specific either to Hiero or to ITGs. We expect that the approach should be applicable with other types of SCFGs, although we note that alignment under SCFGs with an arbitrary number of nonterminals can be NP-hard (Satta and Peserico 2005).

5. HiPDT Two-Pass Translation Architecture and Experiments

The previous complexity analysis suggests that PDAs should excel when used with large translation grammars and relatively small n-gram language models. In hierarchical phrase-based translation, this is a somewhat unusual scenario: it is far more typical that translation tasks which require a large translation grammar also require large language models. To accommodate these requirements we have developed a two-pass decoding strategy in which a weak version of a large language model is applied prior to the expansion of the PDA, after which the full language model is applied to the resulting WFSA in a rescoring pass. An effective way of generating weak language models is by means of entropy pruning under a threshold θ; these are the language models M_1^θ of Section 4.1. Such a two-pass strategy is widely-used in automatic speech recognition (Ljolje, Pereira, and Riley 1999). The steps in two-pass translation using entropy-pruned language models are given here, and depicted in Figure 14.

Step 1. We translate with M_1^θ and G using the same parameters obtained by MERT for the baseline system, with the exception that the word penalty parameter is adjusted to produce hypotheses of roughly the correct length. This produces translation lattices that contain hypotheses with exact scores under G and M_1^θ: $\Pi_2(\{s\} \circ \mathcal{G}) \circ M_1^\theta$.

Step 2. These translation lattices are pruned at beamwidth β: $[\Pi_2(\{s\} \circ \mathcal{G}) \circ M_1^\theta]_\beta$.

Step 3. We remove the M_1^θ scores from the pruned translation lattices, reapply the full language model M_1, and restore the word penalty parameter to the baseline value obtained by MERT. This gives an approximation to $\Pi_2(\{s\} \circ \mathcal{G}) \circ M_1$: scores are correctly assigned under G and M_1, but only hypotheses that survived pruning at Step 2 are included.

We can rescore the lattices produced by the baseline system or by the two-pass system with the larger language model M_2. If $\beta = \infty$ or if $\theta = 0$, the translation lattices obtained in Step 3 should be identical to lattices produced by the baseline system (i.e. the rescoring step is no longer needed). The aim is to increase θ to shrink the language model used at Step 1, but β will then have to increase accordingly to avoid pruning away desirable hypotheses in Step 2.

5.1 Efficient Removal of First-Pass Language Model Scores Using Lexicographic Semirings

The two-pass translation procedure requires removal of the weak language model scores used in the initial expansion of the translation search space; this is done so that only the translation scores under G remain after pruning. In the tropical semiring, the weak LM scores can be 'subtracted' at the path level from the lattice, but this involves a determinization of an unweighted translation lattice, which can be very inefficient.

As an alternative we can define a lexicographic semiring (Shafran et al. 2011; Roark, Sproat, and Shafran 2011), $\langle w_1, w_2 \rangle$ over the tropical weights w_1 and w_2 with the operations $\oplus$ and $\otimes$:

$$\langle w_1, w_2 \rangle \oplus \langle w_3, w_4 \rangle = \begin{cases} \langle w_1, w_2 \rangle & \text{if } w_1 < w_3 \text{ or } (w_1 = w_3 \text{ and } w_2 < w_4) \\ \langle w_3, w_4 \rangle & \text{otherwise} \end{cases} \tag{8}$$

$$\langle w_1, w_2 \rangle \otimes \langle w_3, w_4 \rangle = \langle w_1 + w_3, w_2 + w_4 \rangle \tag{9}$$

The PDA algorithms described in Section 3 are valid under this new semiring since it is commutative and has the path property. In particular, the PDA representing $\{s\} \circ \mathcal{G}$ is constructed so that
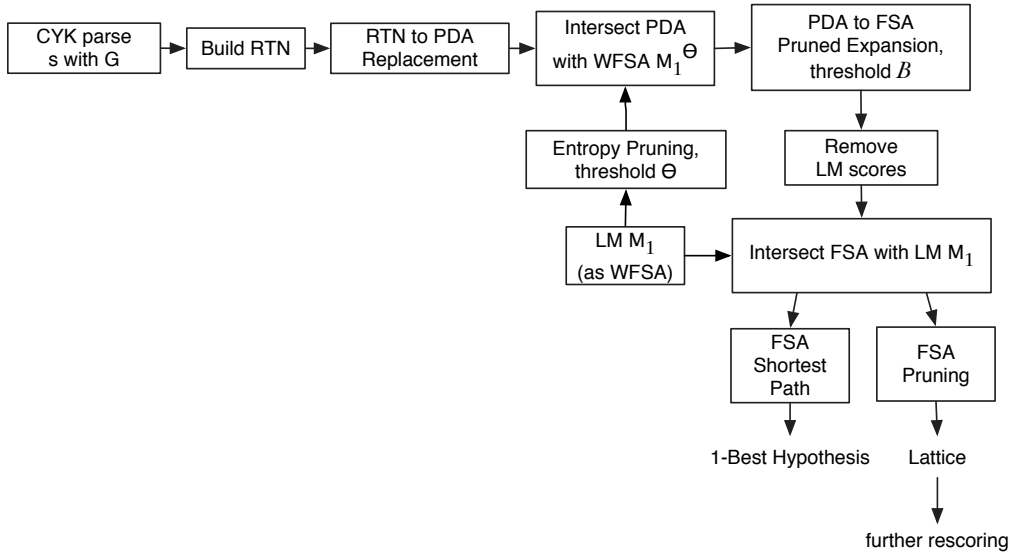
Figure 14
Two-Pass HiPDT Translation with an Entropy Pruned Language Model.

the translation grammar score appears in both w_1 and w_2, i.e. it is duplicated. In the first-pass language model, w_1 has the n-gram language model scores and the w_2 are 0. After composition, the resulting automata have the combined translation grammar score and language model score in the first dimension, and the second dimension contains the translation grammar scores alone. Pruning can be performed under the lexicographic semiring with a threshold set so that only the combined scores in the first dimension are considered. The resulting automata can easily be mapped back into the regular tropical semiring such that only the translation scores in the second dimension are retained (this is a linear operation done by the fstmap operation in the OpenFST library).

5.2 Translation Quality and Modelling Errors in Two-Pass Decoding

We wish to analyze the degree to which the two-pass decoding strategy introduces 'modelling errors' into translation. A modelling error occurs in two-pass decoding whenever the decoder produces a translation whose score is less than the best attainable under the grammar and language model, i.e. whenever the best possible translation is discarded by pruning at Step 2. We refer to these as modelling errors, rather than search errors, because they are due to differences in scores assigned by the models M_1 and M_1^θ.

Ideally, we would compare the two-pass translation system against a baseline system that performs exact translation, without pruning in search, under the grammar G and language model M_1. This would allow us to address the following questions:

- Is a two-pass decoding procedure that uses entropy-pruned language models adequate for translation? How many modelling errors are introduced? Does two-pass decoding impact on translation quality?
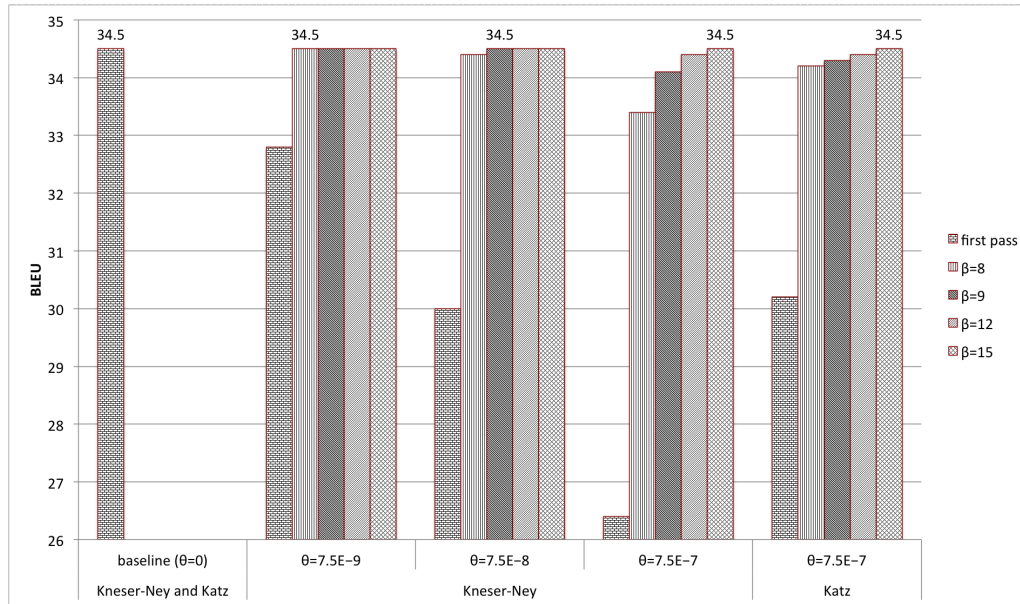
Figure 15
Results (lower case IBM BLEU scores over *test-nw*) under G_{small} with various M_1^θ as obtained with several values of θ. Performance in subsequent rescoring with M_1 after likelihood-based pruning of the resulting translation lattices for various β is also reported. In the pipeline, M_1 (and M_1^θ) are estimated with either Katz or Kneser-Ney smoothing.

- Which smoothing/discounting technique is best suited for the first-pass language model in two-pass translation, and which smoothing/discounting technique is best at avoiding modelling errors?

Our grammar G is not suitable for these experiments, as we do not have a system capable of exact decoding under both G and M_1. To create a suitable baseline we therefore reduce G by excluding rules that have a forward translation probability $p < 0.01$, and refer to this reduced grammar as G_{small}. This process reduces the number of strictly hierarchical rules that apply to our *tune-nw* set from 511K to 189K, while the number of standard phrases is unchanged.

Under G_{small}, both HiFST and HiPDT are able to exactly compose the entire space of possible candidate hypotheses with the language model and to extract the shortest path hypothesis. Because an exact decoding baseline is thus available, we can empirically evaluate the proposed two-pass strategy. Any degradation in translation quality can only be due to the modelling errors introduced by pruning under β with respect to the entropy-pruned M_1^θ.

Figure 15 shows translation performance under grammar G_{small} for different values of entropy pruning threshold θ. Performance is reported after first-pass decoding with M_1^θ (Step 1, Section 5), and after rescoring with M_1 (Step 3, Section 5) the first-pass lattices pruned at alternative β beams. The first column reports the baseline for either Kneser-Ney and Katz language models, which are found by translation without entropy pruning, i.e. performed with M_1. Both yield 34.5 on *test-nw*.

The first and main conclusion from this figure is that the two-pass strategy is adequate because we are always able to recover the baseline performance. As expected, the harsher the entropy-pruning of M_1 (as we lower θ) the greater β must be to recover from the significant degradation in first-pass decoding. But even at a harsh $\theta = 7.5 \times 10^{-7}$, when first-pass perfor-

β	Kneser-Ney	Katz
8	814	619
12	343	212
15	240	110

Table 5
Two-pass translation modelling errors as a function of RTN expansion pruning threshold β. A modelling error occurs whenever the score of a hypothesis produced by the two-pass translation differs from the score found by the exact baseline system. Errors are tabulated over systems reported in Figure 15, at $\theta = 7.5 \times 10^{-7}$.

mance drops over 7 BLEU points, a relatively-low value of $\beta = 15$ can recover the baseline performance.

While the above is true independently of the LM smoothing approach, a second conclusion from the figure is that the choice of LM smoothing does impact first-pass translation performance. For entropy pruning at $\theta = 7.5 \times 10^{-7}$, the Katz LMs perform better for smaller beamwidths β. These results are consistent with the test set perplexities of the entropy pruned LMs (Table 2), and are also in line with other studies of Kneser-Ney smoothing and entropy pruning (Chelba et al. 2010; Roark, Allauzen, and Riley 2013).

Modelling errors are reported in Table 5 at the entropy pruning threshold $\theta = 7.5 \times 10^{-7}$. As expected, modelling errors decrease as the beamwidth β increases, although we find that the language model with Katz smoothing has fewer modelling errors. However modelling errors do not necessarily impact corpus level BLEU scores. For wide beamwidths, e.g. $\beta = 15$ here, there are still some modelling errors, but these are either few enough or subtle enough that two-pass decoding under either smoothing method yields the same corpus level BLEU score as the exact decoding baseline.

5.3 HIPDT Two-Pass Decoding Speed and Translation Performance

- What are the speed and quality tradeoffs for HiPDT as a function of first-pass LM size and translation grammar complexity? How do these compare against the predicted computational complexity?

In this section we turn back to the original large grammar, for which HiFST cannot perform exact decoding (see Table 3). In contrast, HiPDT is able to do exact decoding so we study tradeoffs in speed and translation performance. The speed of two-pass decoding can be increased by decreasing β and/or increasing θ, but at the risk of degradation in translation performance. For grammar G and language model M_1 we plot in Figure 16 the BLEU score against speed as a function of β for a selection of θ values. BLEU score is measured over the entire test set *test-nw* but speed is calculated only on sentences of length up to 20 words (~ 500 sentences). In computing speed we measure not only the PDA operations, but the entire HiPDT decoding process described in Figure 14, including CYK parsing and the application of M_1. We note in passing that these unusually slow decoding speeds are a consequence of the large grammars, language models and broad pruning thresholds chosen for these experiments; in practice, translation with either HiPDT or HiFST is much faster.

In these experiments we find that the language model entropy pruning threshold θ and the likelihood beamwidth β work together to balance speed against translation quality. For every entropy pruning threshold θ value considered, there is a value of β for which there is no
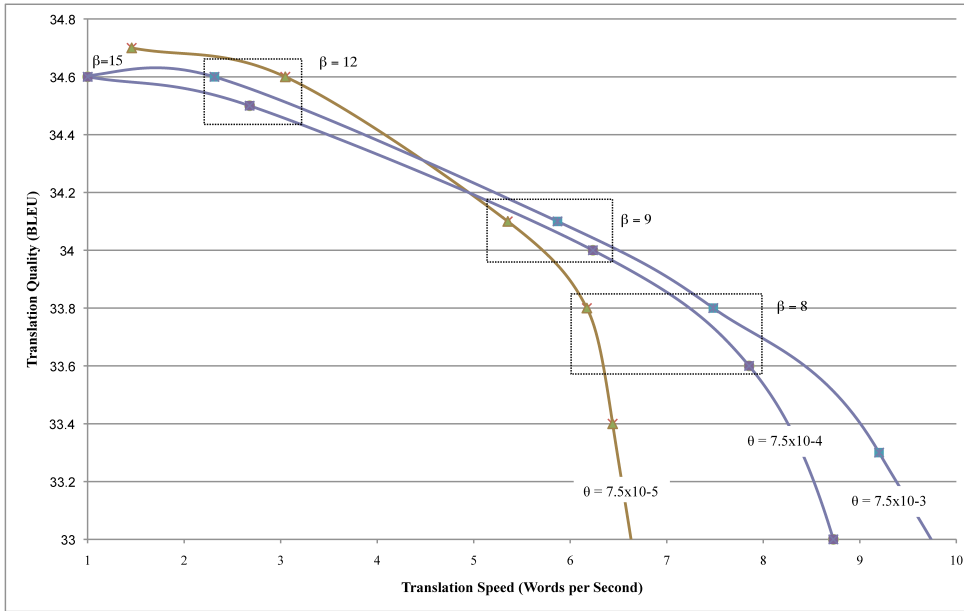
Figure 16
HIPDT translation quality versus speed (decoding with G, M_1^θ + rescoring with M_1) under different entropy pruning thresholds θ and for likelihood beamwidths $\beta = 15, 12, 9, 8, 7$.

degradation in translation quality. For example, suppose we want to attain a translation quality of 34.5 BLEU: then β should be set to 12 or greater. If the goal is to find the fastest system at this level, then we choose $\theta = 7.5 \times 10^{-5}$.

The interaction between pruning in expansion and pruning of the language model is explained by Figure 17, where decoding and rescoring times are shown for various values of θ and β that achieve at least the translation quality target of 34.5. As θ increases, decoding time decreases because a smaller language model is easier to apply; however rescoring times increase, because the larger values of β lead to larger WFSAs after expansion, and these are costly to rescore. The balance occurs at $\theta = 7.5 \times 10^{-5}$ and a translation rate of 3.0 words/sec. In this case, entropy pruning yields a severely shrunken bigram language model, but this may vary depending on the translation grammar and the original, unpruned LM.

5.4 Rescoring with 5-gram Language Models and LMBR Decoding

- Does the HiPDT two-pass decoding generate lattices that can be useful in rescoring?

We now report on rescoring experiments using WFSAs produced by the two-pass HiPDT translation system under the large translation grammar G. We demonstrate that HiPDT can be used to generate large, compact representations of the translation space which are suitable for rescoring with large language models or by alternative decoding procedures. We investigate translation performance by applying versions of the language model M_2 estimated with stupid backoff. We also investigate minimum Bayes risk decoding (Kumar and Byrne 2004) as alterna-

tive search strategy. We are particularly interested in lattice MBR (LMBR) (Tromble et al. 2008) which is well suited for the large WFSAs that the system can generate; we use the implementation described by Blackwood et al. (2010). There are two parameters to be tuned: a scaling parameter to normalize the evidence scores and a word penalty applied to the hypotheses space, and these are tuned jointly on the *tune-nw* set. Results are reported in Figure 18.
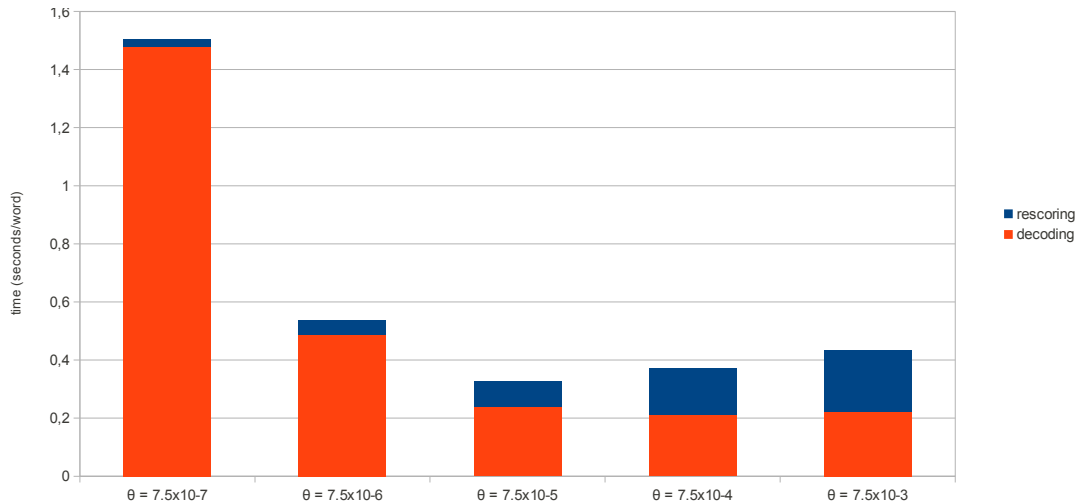
Figure 17
Accumulated decoding+rescoring times for HIPDT under different entropy pruning thresholds, reaching a performance of at least 34.5 BLEU, for which β is set to 12.

We note first that rescoring with the large language model M_2, which is effectively interpolated with M_1, gives consistent gains over initial results obtained with M_1 alone. After 5-gram rescoring there is already +0.5 BLEU improvement compared to G_{small}. With a richer translation grammar we have generated a richer lattice that allows gains to be gotten by our lattice rescoring techniques.

We also find that BLEU scores degrade smoothly as β decreases and the expansion pruning beamwidth narrows, and at all values of β LMBR gives improvement over the MAP hypotheses. Because LMBR relies on posterior distributions over n-grams, we conclude that HiPDT is able to generate compact representations of large search spaces with posteriors that are robust to pruning conditions.

Finally, we find that increasing θ degrades performance quite smoothly for $\beta \geq 9$. Again, with appropriate choices of θ and β we can easily reach a compromise between decoding speed and final performance of our HiPDT system. For instance, with $\theta = 7.5 \times 10^{-7}$ and $\beta = 12$, for which we decode at a rate of 3 words/second as seen in Figure 16, we are losing only 0.5 BLEU after LMBR compared to $\theta = 7.5 \times 10^{-7}$ and $\beta = 15$.

6. 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 the left-to-right nature of n-gram language models.
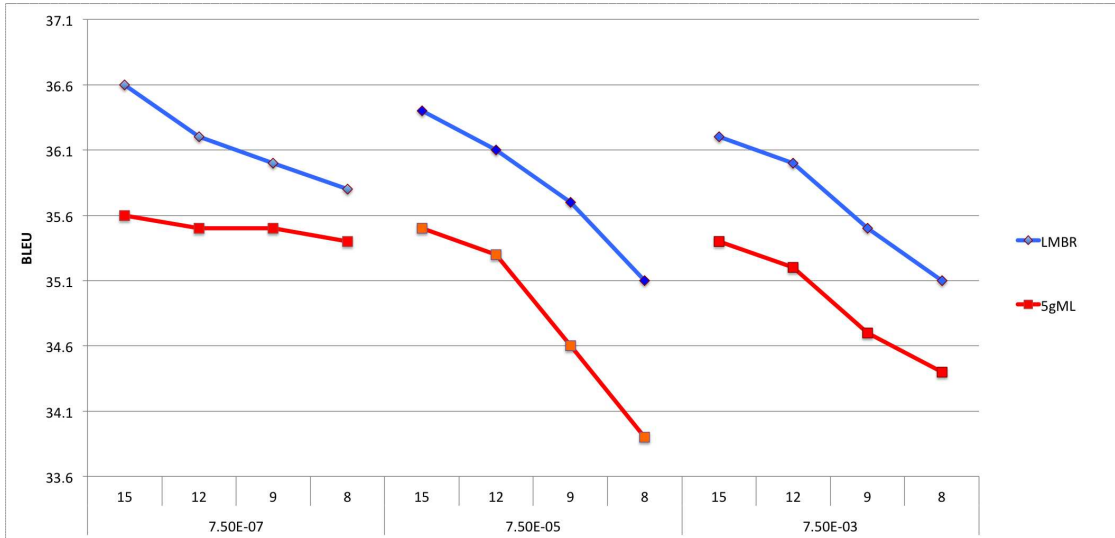
Figure 18
HiPDT decoding with G. Decoding language model M_1^θ and first pass rescoring language model M_1 are Katz. Results on *test-nw* are given for ML-Decoding under the 5-gram stupid backoff language model ('5gML') and for LMBR and for LMBR decoding. Parameter values are $\beta = 15, 12, 9, 8$ and $\theta = 7.5 \times 10^{-7}, 7.5 \times 10^{-5}, 7.5 \times 10^{-3}$.

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, Deng, and Byrne 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). The effect of search errors has also been studied in phrase-based translation by Zens and Ney (2008). Relaxation techniques have also recently been shown to finding exact solutions in parsing (Koo et al. 2010), phrase-based SMT (Chang and Collins 2011), and in tree-to-string translation under trigram language models (Rush and Collins 2011); this prior work involved much smaller grammars and languages models than have been considered here.

Efficiency in synchronous parsing with Hiero grammars and hypergraphs has been studied previously by Dyer (Dyer 2010b), who showed that a single synchronous parsing algorithms (Wu 1997) can be significantly improved upon in practice through hypergraph compositions. We developed similar procedures for our HiFST decoder (Iglesias et al. 2009a; de Gispert et al. 2010) via a different route, after noting that with the space of translations represented as WFSAs, alignment can be performed using operations over WFSTs (Kumar and Byrne 2005).

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, Pereira, and Riley 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.

7. Conclusion

In this paper, we have described a novel approach to hierarchical machine translation using pushdown automata. We have presented fundamental PDA algorithms including composition, shortest-path, (pruned) expansion and replacement and have shown how these can be used in PDA-based MT decoding and how this relates and compares to hypergraph and FSA-based decoding.

On the basis of the experimental results presented in the previous sections, we can now address the questions laid out in Sections 4 and 5:

- A two-pass translation decoding procedure in which translation is first performed with a weak entropy-pruned language model and followed by admissible likelihood-based pruning and rescoring with a full language model can yield good quality translations. Translation performance does not degrade significantly unless the first-pass language model is very heavily pruned.

- As predicted by the analysis of algorithmic complexity, intersection and expansion algorithms based on the PDA representation are able to perform exact decoding with large translation and weak language models. By contrast, RTN to FSA expansion fails with large translation grammars, regardless of the size of the language model. With large translation grammars, language model composition prior to expansion may be more attractive than expansion prior to language model composition.

- Our experimental results suggest that for a translation grammar and a language model of a particular size, and given a value of language model entropy pruning threshold θ, there is a value of the pruned expansion parameter β for which there is no degradation in translation quality with HiPDT. This makes exact decoding under large translation grammars possible. The values of θ and β will be grammar and task dependent.

- While there is some interaction between parameter tuning, pruning thresholds, and language modelling strategies, the variation is not significant enough to indicate that a particular language model or smoothing technique is best. This is particularly true if Minimum Bayes-Risk decoding is applied to the output translation lattices.

Several questions naturally arise about the decoding strategies presented here. One is whether inadmissible pruning methods can be applied to the PDA-based systems that are analogous to those used in current hypergraph-based systems such as cube-pruning (Chiang 2007). Another is whether a hybrid PDA-FSA system, where some parts of the PDA are pre-expanded and some not, could provide benefits over full pre-expansion (FSA) or none (PDA). We leave these for future work.

8. Acknowledgments

Appendix

1. Composition of a weighted PDT and a weighted FST

Given a pair (T_1, T_2) where T_1 is a weighted pushdown transducer and the T_2 is a weighted finite-state transducer, and such that T_1 has input and output alphabets Σ and Δ and T_2 has input and output alphabets Δ and Γ, then there exists a weighted pushdown transducer $T_1 \circ T_2$, which is the *composition* of T_1 and T_2, such that for all $(x, y) \in \Sigma^* \times \Gamma^*$:

$$T = (T_1 \circ T_2)(x, y) = \min_{z \in \Delta^*} (T_1(x, z) + T_2(z, y)) \tag{A.10}$$

We also assume that T_2 has no input-ϵ transitions, noting that for T_2 with input-ϵ transitions, an epsilon filter (Mohri 2009; Allauzen, Riley, and Schalkwyk 2011) generalized to handle parentheses could be used.

A state in T is a pair (q_1, q_2) where q_1 is a state of T_1 and q_2 a state of T_2. Given a transition $e_1 = (q_1, a, b, w_1, q_1')$ in T_1, transitions out of (q_1, q_2) in T are obtained using the following rules. If $b \in \Delta$, then e_1 can be matched with a transition (q_2, b, c, w_2, q_2') in T_2 resulting a transition $((q_1, q_2), a, c, w_1 + w_2, (q_1', q_2'))$ in T. If $b = \epsilon$, then e_1 is matched with staying in q_2 resulting in a transition $((q_1, q_2), a, \epsilon, w_1, (q_1', q_2))$. Finally, if $b = a \in \widehat{\Pi}$, e_1 is also matched with staying in q_2, resulting in a transition $((q_1, q_2), a, a, w_1, (q_1', q_2))$ in T. The initial state is (I_1, I_2) and a state (q_1, q_2) in T is final when both q_1 and q_2 are both final. Weight values are assigned as $\rho((q_1, q_2)) = \rho_1(q_1) + \rho_2(q_2)$.

2. Pruned expansion

Let d^R and B^R be the data-structures computed by the shortest-distance algorithm applied to T^R. For a state q in T' (or equivalently T_β'), let $d[q]$ denote the shortest distance from the initial state to q, $\overline{d}[q]$ the shortest distance from q to the final state and $s[q]$ the destination state of the last unbalanced open-parenthesis transition on a shortest-path from the initial state to q.

The algorithm is based on the following property: letting e denote a transition in T' such that $p[e] = (q, z)$ and $z = z'a$, the weight of a shortest path through e can be expressed as:

$$d[(q, z)] + w[e] + \min_{e' \in B^R[q_s, a]} d^R[n[e], p[e']] + w[e'] + \overline{d}[(n[e'], z')] \tag{B.11}$$

where $(q_s, z) = s[(q, z)]$. This implies that assuming when (q, z) is visited, $d[(n[e'], z')]$ is known, we then have all the required information for deciding whether e should be pruned or retained. In order to ensure that each state is visited once, we need to ensure that $\overline{d}[(q, z)]$ is known when (q, z) is visited so we can apply an A^* queue discipline among the states sharing the same stack.

Both conditions can be achieved by using a queue discipline defined by a partial order $\prec$ such that

$$z \text{ is a prefix of } z' \Rightarrow (q, z) \prec (q', z') \tag{B.12}$$

$$d[(q, z)] + \overline{d}[(q, z)] < d[(q', z)] + \overline{d}[(q', z)] \Rightarrow (q, z) \prec (q', z). \tag{B.13}$$

We also assume that all states sharing the same stack will be dequeued consecutively ($z \neq z' \rightarrow$ for all (q, q'), $(q, z) \prec (q', z')$ or for all (q, q'), $(q', z') \prec (q, z)$). This allows to cache some computations (the D data structure as described below).

PRUNEDEXPANSION(T, β)
1   $(d^R, B^R) \leftarrow$ SHORTESTDISTANCE(T^R)
2   $\lambda \leftarrow d^R[I, f] + \beta$ ▷ *Compute the pruning threshold*
3   $B \leftarrow$ REVERSE(B^R)   ▷ *Compute the balance information in T from the one in T^R*
4   $(I', f') \leftarrow ((I, \epsilon), (f, \epsilon))$ ▷ *I' and f' are the initial and final states of the pruned expansion*
5   $(F', \rho'(f')) \leftarrow (\{f'\}, 0)$
6   $(d[I'], s[I']) \leftarrow (0, I')$
7   $(\overline{d}[I'], \overline{d}[f']) \leftarrow (d^R[I, f], 0)$
8   $(z_D, D[f]) \leftarrow (\epsilon, 0)$
9   $\mathcal{S} \leftarrow Q' \leftarrow \{I'\}$
10 **while** $\mathcal{S} \neq \emptyset$ **do**
11    $(q, z) \leftarrow$ HEAD($\mathcal{S}$)
12   DEQUEUE($\mathcal{S}$)
13   **if** $s[(q, z)] = (q, z)$ **then**
14   **if** $z \neq z_D$ **then** ▷ *If the stack has changed, D needs to be cleared and recomputed*
15   CLEAR(D)
16        $z_D \leftarrow z$
17   **for** each $e \in B[q, z_{|z|}]$ **do**   ▷ *For each close paren. transition balancing the incoming $z_{|z|}$-labeled open paren. transition in q*
18        $D[p[e]] \leftarrow \min(D[p[e]], w[e] + \overline{d}[(n[e], z_1 \cdots z_{|z|-1})])$
19   **for** each $e \in E[q]$ **do**
20   **if** $i[e] \in \Sigma \cup \{\epsilon\}$ **then** ▷ *If $i[e]$ is a regular symbol*
21   **if** RETAINPATH($q, z, w[e], n[e]$) **then**
22          $E' \leftarrow E' \cup \{((q, z), i[e], o[e], w[e], (n[e], z))\}$
23   **elseif** $i[e] \in \Pi$ **then**   ▷ *If $i[e]$ is an open parenthesis*
24        $z' \leftarrow zi[e]$
25        $r \leftarrow$ *false*
26   **for** each $e' \in B[n[e], i[e]]$ **do** ▷ *For each close paren. transition e' that balances e*
27          $w \leftarrow w[e] + d^R[n[e], p[e']] + w[e']$ ▷ *w: weight of the shortest bal. path beginning by e and ending by e' in T*
28          $r \leftarrow r \vee$ RETAINPATH($q, z, w, n[e']$) ▷ *Does the expansion of that path belong to an accepting path below threshold?*
29          $w_F \leftarrow \min(w_F, d^R[n[e], p[e']] + w[e'] + \overline{d}[(n[e], z)])$
30   **if** r **then**   ▷ *If any of the paths considered above are below threshold*
31          $E' \leftarrow E' \cup \{((q, z), \epsilon, \epsilon, w[e], (n[e], z'))\}$
32   PROCESSSTATE($(n[e], z')$)
33          $s[(n[e], z')] \leftarrow (n[e], z')$
34          $d[(n[e], z')] \leftarrow \min(d[(n[e], z')], d[(q, z)] + w[e])$
35          $\overline{d}[(n[e], z')] \leftarrow \min(\overline{d}[(n[e], z')], w_F)$
36   **elseif** $i[e] \in \overline{\Pi}$ and $c_\Pi(zi[e]) \in \Pi^*$ **then**   ▷ *If $i[e]$ is the close parenthesis matching the top of the stack*
37        $z' \leftarrow c_\Pi(zi[e])$
38   **if** $d[(q, z)] + w[e] + \overline{d}[(n[e], z')] \leq \lambda$ **then**
39          $E' \leftarrow E' \cup \{((q, z), \epsilon, \epsilon, w[e], (n[e], z'))\}$
40 **return** $(\Sigma, \Delta, \Pi, \overline{\Pi}, Q', E', I', F', \rho')$

RETAINPATH(q, z, w, q')
1   ▷ *Returns true iff a path from (q, z) to (q', z) with weight w belongs to an accepting path below threshold*
2   $w_I \leftarrow d[(q, z)] + w$ ▷ *Shortest distance from I to (q', z) when taking a path form (q, z) to (q', z) of weight w*
3   $w_F \leftarrow \min\{d^R[q', t] + D[t] | D[t] \neq \infty\}$ ▷ *Current estimate of s. d. from (q', z) to f'*
4   **if** $w_I < d[(q', z)]$ **then**   ▷ *If w_I is a better estimate of s.-d. from I' to (q', z), update $d[(q', z)]$ and $s[(q', z)]$*
5    $d[(q', z)] \leftarrow w_I$
6    $s[(q', z)] \leftarrow s[(q, z)]$
7   **if** $w_F < \overline{d}[(q', z)]$ **then**   ▷ *If w_F is a better estimate of s. d. from (q', z) to f', update $\overline{d}[(q', z)]$*
8    $\overline{d}[(q', z)] \leftarrow w_F$
9   **if** $\lambda < w_I + w_F$ **then** ▷ *$w_I + w_F$: min. weight of an accepting path taking a path of weight w from (q, z) to (q', z)*
10   **return** *false*
11 PROCESSSTATE((q', z))
12 **return** $true$

PROCESSSTATE((q, z))
1   **if** $(q, z) \notin Q'$ **then**   ▷ *If state (q, z) does not exists yet, create it and add it to the queue*
2    $Q' \leftarrow Q' \cup \{(q, z)\}$
3   ENQUEUE($\mathcal{S}, (q, z)$)

Figure 19
PDT pruned expansion algorithm. We assume that $F = \{f\}$ and $\rho(f) = 0$ to simplify the presentation.

The pseudo code of the algorithm is given in Figure 19. First, the shortest distance algorithm is applied to T^R and the absolute pruning threshold is computed accordingly (lines 1-2). The resulting balanced data information is then reversed (line 3). The initial and final states are created (lines 4-5) and the d, $\overline{d}$ and D data structures are initialized accordingly (lines 6-8). The default value in these data structures is assumed to be ∞. The queue is initialized containing the initial state (line 9).

The state (q, z) at the head of queue in dequeued (lines 10-12). If (q, z) admits an incoming open-parenthesis transition, B contains the balance information for that state and D can be updated accordingly (lines 13-18).

If e is a regular transition, the resulting transition $((q, z), i[e], o[e], w[e], (n[e], z))$ in T' can be pruned using the criterion derived from (B.11). If it is retained, the transition is created as well as its destination state $(n[e], z)$ if needed (lines 20-22).

If e is an open-parenthesis transition, each balanced path starting by the resulting transition in T' and ending by a close-parenthesis transition is treated as a meta-transition and pruned using the same criterion as regular transitions (lines 23-29). If any of these meta-transitions is retained, the transition $((q, z), \epsilon, \epsilon, w[e], (n[e], zi[e]))$ resulting from e is created as well as its destination state $(n[e], zi[e])$ if needed (lines 30-35).

If e is a closed-parenthesis transition, it is created if it belongs to a balanced path below the threshold (line 36-39).

Finally, the resulting transducer T'_β is returned (line 40).

References

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

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

Allauzen, Cyril, 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.

Allauzen, Cyril, 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.

Bar-Hillel, Y., 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*. Addison-Wesley, pages 116–150.

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

Blackwood, Graeme, Adrià de Gispert, and William Byrne. 2010. Efficient path counting transducers for minimum bayes-risk decoding of statistical machine translation lattices. In *Proceedings of the ACL: Short Papers*, pages 27–32.

Brants, Thorsten, 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.

Chang, Yin-Wen and Michael Collins. 2011. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of EMNLP*, pages 26–37.

Chelba, Ciprian, 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.

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

de Gispert, Adrià, 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):201–228.

Deng, Yonggang 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.

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

Dyer, Chris. 2010a. *A Formal Model of Ambiguity and its Applications in Machine Translation*. Ph.D. thesis, University of Maryland.

Dyer, Chris. 2010b. Two monolingual parses are better than one (synchronous parse). In *Proceedings of NAACL-HLT*, pages 263–266.

Galley, M., M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a translation rule. In

Proceedings of HLT-NAACL, pages 273–280.

Hopkins, M. and G. Langmead. 2010. SCFG decoding without binarization. In *Proceedings of EMNLP*, pages 646–655.

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

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

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

Huang, Liang, 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.

Iglesias, Gonzalo, 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.

Iglesias, Gonzalo, 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.

Katz, Slava M. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Kneser, Reinhard and Herman Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of ICASSP*, volume 1, pages 181–184.

Koo, Terry, 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.

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

Kumar, Shankar and William Byrne. 2004. Minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of HLT-NAACL*, pages 169–176.

Kumar, Shankar and William Byrne. 2005. Local phrase reordering models for statistical machine translation. In *Proceedings of EMNLP-HLT*, pages 161–168.

Kumar, Shankar, 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.

Lang, Bernard. 1974. Deterministic techniques for efficient non-deterministic parsers. In *Proceedings of ICALP*, pages 255–269.

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

Mohri, Mehryar. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7:321–350.

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

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

Nederhof, Mark-Jan and Giorgio Satta. 2006. Probabilistic parsing strategies. *Journal of the ACM*, 53(3):406–436.

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

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

Prasad, R., 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.

Roark, Brian, Cyril Allauzen, and Michael Riley. 2013. Smoothed marginal distribution constraints for language modeling. In *Proceedings of ACL*, pages 43–52.

Roark, Brian, Richard Sproat, and Izhak Shafran. 2011. Lexicographic semirings for exact automata encoding of sequence models. In *Proceedings of ACL-HLT*, pages 1–5.

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

Satta, Giorgio and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 803–810, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Shafran, Izhak, Richard Sproat, Mahsa Yarmohammadi, and Brian Roark. 2011. Efficient determinization of tagged word

lattices using categorial and lexicographic semirings. In *Proceedings of ASRU*.

Stolcke, Andreas. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

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

Tromble, Roy, Shankar Kumar, Franz J. Och, and Wolfgang Macherey. 2008. Lattice Minimum Bayes-Risk decoding for statistical machine translation. In *Proceedings of EMNLP*, pages 620–629.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–403.

Xiao, Tong, Mu Li, Dongdong Zhang, Jingbo Zhu, and Ming Zhou. 2009. Better synchronous binarization for machine translation. In *Proceedings of EMNLP*, pages 362–370.

Zens, Richard and Hermann Ney. 2008. Improvements in dynamic programming beam search for phrase-based statistical machine translation. In *Proceedings of IWSLT*, pages 195–205.

Zhang, Hao, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of HLT-NAACL*, pages 256–263.

Zollmann, Andreas and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of NAACL Workshop on Statistical Machine Translation*, pages 138–141.