
Proceedings Paper:
Aziz, W., Dymetman, M. and Specia, L. orcid.org/0000-0002-5495-3128 (2014) Exact decoding for phrase-based statistical machine translation. In: EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference. EMNLP 2014: Conference on Empirical Methods in Natural Language Processing, 25th –29th October 2014, Doha, Qatar. , pp. 1237-1249. ISBN 9781937284961

Exact Decoding for Phrase-Based Statistical Machine Translation

Wilker Aziz[†] **Marc Dymetman**[‡] **Lucia Specia**[†]

[†]Department of Computer Science, University of Sheffield, UK
W.Aziz@sheffield.ac.uk
L.Specia@sheffield.ac.uk
[‡]Xerox Research Centre Europe, Grenoble, France
Marc.Dymetman@xrce.xerox.com

Abstract

The combinatorial space of translation derivations in phrase-based statistical machine translation is given by the intersection between a translation lattice and a target language model. We replace this intractable intersection by a tractable relaxation which incorporates a low-order upperbound on the language model. Exact optimisation is achieved through a coarse-to-fine strategy with connections to adaptive rejection sampling. We perform exact optimisation with unpruned language models of order 3 to 5 and show search-error curves for beam search and cube pruning on standard test sets. This is the first work to tractably tackle exact optimisation with language models of orders higher than 3.

1 Introduction

In Statistical Machine Translation (SMT), the task of producing a translation for an input string $\mathbf{x} = \langle x_1, x_2, \ldots, x_I \rangle$ is typically associated with finding the best derivation $\mathbf{d}^*$ compatible with the input under a linear model. In this view, a derivation is a structured output that represents a sequence of steps that covers the input producing a translation. Equation 1 illustrates this *decoding* process.

$$\mathbf{d}^* = \underset{\mathbf{d} \in \mathcal{D}(\mathbf{x})}{\operatorname{argmax}} f(\mathbf{d}) \tag{1}$$

The set $\mathcal{D}(\mathbf{x})$ is the space of all derivations compatible with $\mathbf{x}$ and supported by a model of translational equivalences (Lopez, 2008). The function $f(\mathbf{d}) = \Lambda \cdot \mathbf{H}(\mathbf{d})$ is a linear parameterisation of the model (Och, 2003). It assigns a real-valued score (or weight) to every derivation $\mathbf{d} \in \mathcal{D}(\mathbf{x})$, where $\Lambda \in \mathbb{R}^m$ assigns a relative importance to different aspects of the derivation independently captured by m feature functions $\mathbf{H}(\mathbf{d}) = \langle H_1(\mathbf{d}), \ldots, H_m(\mathbf{d}) \rangle \in \mathbb{R}^m$.

The fully parameterised model can be seen as a discrete weighted set such that feature functions factorise over the steps in a derivation. That is, $H_k(\mathbf{d}) = \sum_{e \in \mathbf{d}} h_k(e)$, where h_k is a (local) feature function that assesses steps independently and $\mathbf{d} = \langle e_1, e_2, \ldots, e_l \rangle$ is a sequence of l steps. Under this assumption, each step is assigned the weight $w(e) = \Lambda \cdot \langle h_1(e), h_2(e), \ldots, h_m(e) \rangle$. The set $\mathcal{D}$ is typically finite, however, it contains a very large number of structures — exponential (or even factorial, see §2) with the size of $\mathbf{x}$ — making exhaustive enumeration prohibitively slow. Only in very restricted cases combinatorial optimisation techniques are directly applicable (Tillmann et al., 1997; Och et al., 2001), thus it is common to resort to heuristic techniques in order to find an approximation to $\mathbf{d}^*$ (Koehn et al., 2003; Chiang, 2007).

Evaluation exercises indicate that approximate search algorithms work well in practice (Bojar et al., 2013). The most popular algorithms provide solutions with unbounded error, thus precisely quantifying their performance requires the development of a tractable exact decoder. To date, most attempts were limited to short sentences and/or somewhat toy models trained with artificially small datasets (Germann et al., 2001; Iglesias et al., 2009; Aziz et al., 2013). Other work has employed less common approximations to the model reducing its search space complexity (Kumar et al., 2006; Chang and Collins, 2011; Rush and Collins, 2011). These do not answer whether or not current decoding algorithms perform well at real translation tasks with state-of-the-art models.

We propose an exact decoder for phrase-based SMT based on a coarse-to-fine search strategy (Dymetman et al., 2012). In a nutshell, we relax the decoding problem with respect to the Language Model (LM) component. This coarse view is incrementally refined based on evidence col-

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1237–1249,
October 25-29, 2014, Doha, Qatar.

lected via maximisation. A refinement increases the complexity of the model only slightly, hence dynamic programming remains feasible throughout the search until convergence. We test our decoding strategy with realistic models using standard data sets. We also contribute with optimum derivations which can be used to assess future improvements to approximate decoders. In the remaining sections we present the general model (§2), survey contributions to exact optimisation (§3), formalise our novel approach (§4), present experiments (§5) and conclude (§6).

2 Phrase-based SMT

In phrase-based SMT (Koehn et al., 2003), the building blocks of translation are pairs of phrases (or *biphrases*). A translation derivation $\mathbf{d}$ is an ordered sequence of *non-overlapping* biphrases which covers the input text in arbitrary order generating the output from left to right.[1]

$$f(\mathbf{d}) = \psi(\mathbf{y}) + \sum_{i=1}^{l} \phi(e_i) + \sum_{i=1}^{l-1} \delta(e_i, e_{i-1}) \quad (2)$$

Equation 2 illustrates a standard phrase-based model (Koehn et al., 2003): ψ is a weighted target n-gram LM component, where $\mathbf{y}$ is the yield of $\mathbf{d}$; ϕ is a linear combination of features that decompose over phrase pairs directly (e.g. backward and forward translation probabilities, lexical smoothing, and word and phrase penalties); and δ is an unlexicalised penalty on the number of skipped input words between two adjacent biphrases. The weighted logic program in Figure 1 specifies the fully parameterised weighted set of solutions, which we denote $\langle \mathcal{D}(\mathbf{x}), f(\mathbf{d}) \rangle$.[2]

A weighted logic program starts from its axioms and follows exhaustively deducing new items by combination of existing ones and no deduction happens twice. In Figure 1, a *nonteminal* item summarises partial derivation (or *hypotheses*). It is denoted by $[C, r, \gamma]$ (also known as *carry*), where: C is a coverage vector, necessary to impose the non-overlapping constraint; r is the rightmost position most recently covered, necessary for the computation of δ; and γ is the last $n-1$ words

$$\begin{array}{ll}
\textsc{Item} & \left[\{0,1\}^I, [0, I+1], \Delta^{n-1}\right] \\
\textsc{Goal} & \left[1^I, I+1, \textsc{eos}\right] \\
\textsc{Axiom} & \\
& \frac{\langle \textsc{bos} \rightarrow \textsc{bos} \rangle}{[0^I, 0, \textsc{bos}] : \psi(\textsc{bos})} \\
\textsc{Expand} & \\
& \frac{\left[C, r, y_{j-n+1}^{j-1}\right] \quad \left\langle x_i^{i'} \xrightarrow{\phi_r} y_j^{j'} \right\rangle}{\left[C', i', y_{j'-n+2}^{j'}\right] : w} \quad \bigoplus_{k=i}^{i'} c_k = \bar{0} \\
& \text{where} \quad c'_k = c_k \text{ if } k < i \text{ or } k > i' \text{ else } \bar{1} \\
& \qquad\quad w = \phi_r \otimes \delta(r, i) \otimes \psi(y_j^{j'} | y_{j-n+1}^{j-1}) \\
\textsc{Accept} & \\
& \frac{\left[1^I, r, \gamma\right]}{[1^I, I+1, \textsc{eos}] : \delta(r, I+1) \otimes \psi(\textsc{eos}|\gamma)} \quad r \leq I
\end{array}$$

Figure 1: Specification for the weighted set of translation derivations in phrase-based SMT with unconstrained reordering.

in the yield, necessary for the LM component. The program expands partial derivations by concatenation with a translation rule $\left\langle x_i^{i'} \xrightarrow{\phi_r} y_j^{j'} \right\rangle$, that is, an instantiated biphrase which covers the span $x_i^{i'}$ and yields $y_j^{j'}$ with weight ϕ_r. The side condition imposes the non-overlapping constraint (c_k is the kth bit in C). The antecedents are used to compute the weight of the deduction, and the carry is updated in the consequent (item below the horizontal line). Finally, the rule ACCEPT incorporates the end-of-sentence boundary to complete items.[3]

It is perhaps illustrative to understand the set of weighted translation derivations as the intersection between two components. One that is only locally parameterised and contains all translation derivations (a translation *lattice* or *forest*), and one that re-ranks the first as a function of the interactions between translation steps. The model of translational equivalences parameterised only with ϕ is an instance of the former. An n-gram LM component is an instance of the latter.

2.1 Hypergraphs

A *backward-hypergraph*, or simply *hypergraph*, is a generalisation of a graph where edges have multiple origins and one destination (Gallo et al., 1993). They can represent both finite-state and context-free weighted sets and they have been widely used in SMT (Huang and Chiang, 2007). A hypergraph is defined by a set of nodes (or ver-

[1] Preventing phrases from overlapping requires an exponential number of constraints (the powerset of $\mathbf{x}$) rendering the problem NP-complete (Knight, 1999).

[2] Weighted logics have been extensively used to describe weighted sets (Lopez, 2009), operations over weighted sets (Chiang, 2007; Dyer and Resnik, 2010), and a variety of dynamic programming algorithms (Cohen et al., 2008).

[3] Figure 1 can be seen as a specification for a weighted acyclic finite-state automaton whose states are indexed by $[l, C, r]$ and transitions are labelled with biphrases. However, for generality of representation, we opt for using acyclic hypergraphs instead of automata (see §2.1).

tices) V and a weighted set of edges $\langle E, w \rangle$. An edge e connects a sequence of nodes in its tail $\mathrm{t}[e] \in V^*$ under a head node $\mathrm{h}[e] \in V$ and has weight $w(e)$. A node v is a *terminal node* if it has no incoming edges, otherwise it is a *nonterminal node*. The node that has no outgoing edges, is called *root*, with no loss of generality we can assume hypergraphs to have a single root node.

Hypergraphs can be seen as instantiated logic programs. In this view, an item is a template for the creation of nodes, and a weighted deduction rule is a template for edges. The tail of an edge is the sequence of nodes associated with the antecedents, and the head is the node associated with the consequent. Even though the space of weighted derivations in phrase-based SMT is finite-state, using a hypergraph as opposed to a finite-state automaton makes it natural to encode multi-word phrases using tails. We opt for representing the target side of the biphrase as a sequence of terminals nodes, each of which represents a target word.

3 Related Work

3.1 Beam filling algorithms

Beam search (Koehn et al., 2003) and cube pruning (Chiang, 2007) are examples of state-of-the-art approximate search algorithms. They approximate the intersection between the translation forest and the language model by expanding a limited beam of hypotheses from each nonterminal node. Hypotheses are organised in priority queues according to common traits and a fast-to-compute heuristic view of *outside weights* (cheapest way to complete a hypothesis) puts them to compete at a fairer level. Beam search exhausts a node's possible expansions, scores them, and discards all but the k highest-scoring ones. This process is wasteful in that k is typically much smaller than the number of possible expansions. Cube pruning employs a priority queue at beam filling and computes k high-scoring expansions directly in near best-first order. The parameter k is known as *beam size* and it controls the time-accuracy trade-off of the algorithm.

Heafield et al. (2013a) move away from using the language model as a black-box and build a more involved beam filling algorithm. Even though they target approximate search, some of their ideas have interesting connections to ours (see §4). They group hypotheses that share partial language model state (Li and Khudanpur, 2008) reasoning over multiple hypotheses at once. They fill a beam in best-first order by iteratively visiting groups using a priority queue: if the top group contains a single hypothesis, the hypothesis is added to the beam, otherwise the group is partitioned and the parts are pushed back to the queue. More recently, Heafield et al. (2014) applied their beam filling algorithm to phrase-based decoding.

3.2 Exact optimisation

Exact optimisation for monotone translation has been done using A* search (Tillmann et al., 1997) and finite-state operations (Kumar et al., 2006). Och et al. (2001) design near-admissible heuristics for A* and decode very short sentences (6-14 words) for a word-based model (Brown et al., 1993) with a maximum distortion strategy ($d = 3$).

Zaslavskiy et al. (2009) frame phrase-based decoding as an instance of a generalised Travelling Salesman Problem (TSP) and rely on robust solvers to perform decoding. In this view, a *salesman graph* encodes the translation options, with each node representing a biphrase. Non-overlapping constraints are imposed by the TSP solver, rather than encoded directly in the salesman graph. They decode only short sentences (17 words on average) using a 2-gram LM due to salesman graphs growing too large.[4]

Chang and Collins (2011) relax phrase-based models w.r.t. the non-overlapping constraints, which are replaced by soft penalties through Lagrangian multipliers, and intersect the LM component exhaustively. They do employ a maximum distortion limit ($d = 4$), thus the problem they tackle is no longer NP-complete. Rush and Collins (2011) relax a hierarchical phrase-based model (Chiang, 2005)[5] w.r.t. the LM component. The translation forest and the language model trade their weights (through Lagrangian multipliers) so as to ensure agreement on what each component believes to be the maximum. In both approaches, when the dual converges to a compliant solution, the solution is guaranteed to be optimal. Other-

[4] Exact decoding had been similarly addressed with Integer Linear Programming (ILP) in the context of word-based models for very short sentences using a 2-gram LM (Germann et al., 2001). Riedel and Clarke (2009) revisit that formulation and employ a cutting-plane algorithm (Dantzig et al., 1954) reaching 30 words.

[5] In hierarchical translation, reordering is governed by a synchronous context-free grammar and the underlying problem is no longer NP-complete. Exact decoding remains infeasible because the intersection between the translation forest and the target LM is prohibitively slow.

wise, a subset of the constraints is explicitly added and the dual optimisation is repeated. They handle sentences above average length, however, resorting to compact rulesets (10 translation options per input segment) and using only 3-gram LMs.

In the context of hierarchical models, Aziz et al. (2013) work with unpruned forests using upperbounds. Their approach is the closest to ours. They also employ a coarse-to-fine strategy with the OS* framework (Dymetman et al., 2012), and investigate unbiased sampling in addition to optimisation. However, they start from a coarser upperbound with unigram probabilities, and their refinement strategies are based on exhaustive intersections with small n-gram matching automata. These refinements make forests grow unmanageable too quickly. Because of that, they only deal with very short sentences (up to 10 words) and even then decoding is very slow. We design better upperbounds and a more efficient refinement strategy. Moreover, we decode long sentences using language models of order 3 to 5.[6]

4 Approach

4.1 Exact optimisation with OS*

Dymetman et al. (2012) introduced OS*, a unified view of optimisation and sampling which can be seen as a cross between adaptive rejection sampling (Robert and Casella, 2004) and A* optimisation (Hart et al., 1968). In this framework, a complex goal distribution is upperbounded by a simpler proposal distribution for which optimisation (and sampling) is feasible. This proposal is incrementally refined to be closer to the goal until the maximum is found (or until the sampling performance exceeds a certain level).

Figure 2 illustrates exact optimisation with OS*. Suppose f is a complex target goal distribution, such that we cannot optimise f, but we can assess $f(\mathbf{d})$ for a given $\mathbf{d}$. Let $g^{(0)}$ be an upperbound to f, i.e., $g^{(0)}(\mathbf{d}) \geq f(\mathbf{d})$ for all $\mathbf{d} \in \mathcal{D}(\mathbf{x})$. Moreover, suppose that $g^{(0)}$ is simple enough to be optimised efficiently. The algorithm proceeds by solving $\mathbf{d}_0 = \text{argmax}_{\mathbf{d}}\, g^{(0)}(\mathbf{d})$ and computing the quantity $r_0 = f(\mathbf{d}_0)/g^{(0)}(\mathbf{d}_0)$. If r_0 were sufficiently close to 1, then $g^{(0)}(\mathbf{d}_0)$ would be sufficiently close to $f(\mathbf{d}_0)$ and we would have found the optimum. However, in the illustration $g^{(0)}(\mathbf{d}_0) \gg f(\mathbf{d}_0)$, thus $r_0 \ll 1$. At this point the algorithm has concrete evidence to motivate a refinement of $g^{(0)}$ that can lower its maximum, bringing it closer to $f^* = \max_{\mathbf{d}} f(\mathbf{d})$ at the cost of some small increase in complexity. The refined proposal must remain an upperbound to f. To continue with the illustration, suppose $g^{(1)}$ is obtained. The process is repeated until eventually $g^{(t)}(\mathbf{d}_t) = f(\mathbf{d}_t)$, where $\mathbf{d}_t = \text{argmax}_{\mathbf{d}}\, g^{(t)}(\mathbf{d})$, for some finite t. At which point $\mathbf{d}_t$ is the optimum derivation $\mathbf{d}^*$ from f and the sequence of upperbounds provides a proof of optimality.[7]

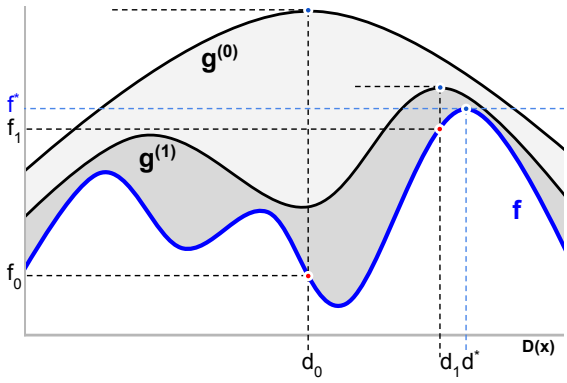

Figure 2: Sequence of incrementally refined upperbound proposals.

4.2 Model

We work with phrase-based models in a standard parameterisation (Equation 2). However, to avoid having to deal with NP-completeness, we constrain reordering to happen only within a limited window given by a notion of *distortion limit*. We require that the last source word covered by any biphrase must be within d words from the leftmost uncovered source position (Lopez, 2009). This is a widely used strategy and it is in use in the Moses toolkit (Koehn et al., 2007).[8]

Nevertheless, the problem of finding the best

[6] The intuition that a full intersection is wasteful is also present in (Petrov et al., 2008) in the context of approximate search. They start from a coarse distribution based on automatic word clustering which is refined in multiple passes. At each pass, hypotheses are pruned a posteriori on the basis of their marginal probabilities, and word clusters are further split. We work with upperbounds, rather than word clusters, with unpruned distributions, and perform exact optimisation.

[7] If $\mathbf{d}$ is a maximum from g and $g(\mathbf{d}) = f(\mathbf{d})$, then it is easy to show by contradiction that $\mathbf{d}$ is the actual maximum from f: if there existed $\mathbf{d}'$ such that $f(\mathbf{d}') > f(\mathbf{d})$, then it follows that $g(\mathbf{d}') \geq f(\mathbf{d}') > f(\mathbf{d}) = g(\mathbf{d})$, and hence $\mathbf{d}$ would not be a maximum for g.

[8] A distortion limit characterises a form of pruning that acts directly in the generative capacity of the model leading to induction errors (Auli et al., 2009). Limiting reordering like that lowers complexity to a polynomial function of I and an exponential function of the distortion limit.

derivation under the model remains impracticable due to nonlocal parameterisation (namely, the n-gram LM component). The weighted set $\langle \mathcal{D}(\mathbf{x}), f(\mathbf{d}) \rangle$, which represents the objective, is a complex hypergraph which we cannot afford to construct. We propose to construct instead a simpler hypergraph for which optimisation by dynamic programming is feasible. This *proxy* represents the weighted set $\langle \mathcal{D}(\mathbf{x}), g^{(0)}(\mathbf{d}) \rangle$, where $g^{(0)}(\mathbf{d}) \geq f(\mathbf{d})$ for every $\mathbf{d} \in \mathcal{D}(\mathbf{x})$. Note that this proposal contains **exactly** the same translation options as in the original decoding problem. The simplification happens only with respect to the parameterisation. Instead of intersecting the complete n-gram LM distribution explicitly, we implicitly intersect a simpler upperbound view of it, where by *simpler* we mean *lower-order*.

$$g^{(0)}(\mathbf{d}) = \sum_{i=1}^{l} \omega(\mathrm{y}[e_i]) + \sum_{i=1}^{l} \phi(e_i) + \sum_{i=1}^{l-1} \delta(e_i, e_{i-1}) \quad (3)$$

Equation 3 shows the model we use as a proxy to perform exact optimisation over f. In comparison to Equation 2, the term $\sum_{i=1}^{l} \omega(\mathrm{y}[e_i])$ replaces $\psi(\mathbf{y}) = \lambda_\psi p_{\text{LM}}(\mathbf{y})$. While ψ weights the yield $\mathbf{y}$ taking into account all n-grams (including those crossing the boundaries of phrases), ω weights edges in isolation. Particularly, $\omega(\mathrm{y}[e_i]) = \lambda_\psi q_{\text{LM}}(\mathrm{y}[e_i])$, where $\mathrm{y}[e_i]$ returns the sequence of target words (a target phrase) associated with the edge, and $q_{\text{LM}}(\cdot)$ is an upperbound on the true LM probability $p_{\text{LM}}(\cdot)$ (see §4.3). It is obvious from Equation 3 that our proxy model is much simpler than the original — the only form of nonlocal parameterisation left is the distortion penalty, which is simple enough to represent exactly.

The program in Figure 3 illustrates the construction of $\langle \mathcal{D}(\mathbf{x}), g^{(0)}(\mathbf{d}) \rangle$. A nonterminal item $[l, C, r]$ stores: the leftmost uncovered position l and a truncated coverage vector C (together they track d input positions); and the rightmost position r most recently translated (necessary for the computation of the distortion penalty). Observe how nonterminal items **do not** store the LM state.[9] The rule ADJACENT expands derivations by concatenation with a biphrase $\left\langle x_i^{i'} \rightarrow y_j^{j'} \right\rangle$ starting at the leftmost uncovered position $i = l$. That causes the coverage window to move ahead to the next leftmost uncovered position: $l' = l + \alpha_1(C) + 1$,

[9] Drawing a parallel to (Heafield et al., 2013a), a nonterminal node in our hypergraph groups derivations while exposing only an empty LM state.

ITEM $\left[[1, I+1], \{0,1\}^{d-1}, [0, I+1]\right]$
GOAL $[I, \emptyset, I+1]$
AXIOMS
$$\frac{\langle \text{BOS} \rightarrow \text{BOS} \rangle}{[1, 0^{d-1}, 0] : \omega(\text{BOS})}$$
ADJACENT
$$\frac{[l, C, r] \quad \left\langle x_i^{i'} \xrightarrow{\phi_r} y_j^{j'} \right\rangle}{[l', C', i'] : \phi_r \otimes \delta(r, i') \otimes \omega(y_j^{j'})} \quad \begin{array}{l} i = l \\ \bigoplus_{k=i-l}^{i'-l} c_k = \bar{0} \end{array}$$
where $l' = l + \alpha_1(C) + 1$, $C' \ll \alpha_1(C) + 1$
NON-ADJACENT
$$\frac{[l, C, r] \quad \left\langle x_i^{i'} \xrightarrow{\phi_r} y_j^{j'} \right\rangle}{[l, C', i'] : \phi_r \otimes \delta(r, i') \otimes \omega(y_j^{j'})} \quad \begin{array}{l} i > l \\ \bigoplus_{k=i-l}^{i'-l} c_k = \bar{0} \\ |r - i + 1| \leq d \\ |i' - l + 1| \leq d \end{array}$$
where $c'_k = c_k$ if $k < i - l$ or $k > i' - l$ else $\bar{1}$
ACCEPT
$$\frac{[I+1, C, r]}{[I+1, \emptyset, I+1] : \delta(r, I+1) \otimes \omega(\text{EOS})} \quad r \leq I$$

Figure 3: Specification of the initial proposal hypergraph. This program allows the same reorderings as (Lopez, 2009) (see logic WLd), however, it does not store LM state information and it uses the upperbound LM distribution $\omega(\cdot)$.

where $\alpha_1(C)$ returns the number of leading 1s in C, and $C' \ll \alpha_1(C) + 1$ represents a left-shift. The rule NON-ADJACENT handles the remaining cases $i > l$ provided that the expansion skips at most d input words $|r - i + 1| \leq d$. In the consequent, the window C is simply updated to record the translation of the input span $i..i'$. In the non-adjacent case, a *gap constraint* imposes that the resulting item will require skipping no more than d positions before the leftmost uncovered word is translated $|i' - l + 1| \leq d$.[10] Finally, note that deductions incorporate the weighted upperbound $\omega(\cdot)$, rather than the true LM component $\psi(\cdot)$.[11]

4.3 LM upperbound and Max-ARPA

Following Carter et al. (2012) we compute an upperbound on n-gram conditional probabilities by precomputing max-backoff weights stored in a "Max-ARPA" table, an extension of the ARPA format (Jurafsky and Martin, 2000).

A standard ARPA table T stores entries

[10] This constraint prevents items from becoming dead-ends where incomplete derivations require a reordering step larger than d. This is known to prevent many search errors in beam search (Chang and Collins, 2011).

[11] Unlike Aziz et al. (2013), rather than unigrams only, we score all n-grams within a translation rule (including incomplete ones).

$\langle \mathtt{Z}, \mathtt{Z.p}, \mathtt{Z.b} \rangle$, where $\mathtt{Z}$ is an n-gram equal to the concatenation $\mathtt{Pz}$ of a prefix $\mathtt{P}$ with a word $\mathtt{z}$, $\mathtt{Z.p}$ is the conditional probability $p(\mathtt{z}|\mathtt{P})$, and $\mathtt{Z.b}$ is a so-called "backoff" weight associated with $\mathtt{Z}$. The conditional probability of an arbitrary n-gram $p(\mathtt{z}|\mathtt{P})$, whether listed or not, can then be recovered from T by the simple recursive procedure shown in Equation 4, where tail deletes the first word of the string $\mathtt{P}$.

$$p(\mathtt{z}|\mathtt{P}) = \begin{cases} p(\mathtt{z}|\,\mathrm{tail}(\mathtt{P})) & \mathtt{Pz} \notin T \text{ and } \mathtt{P} \notin T \\ p(\mathtt{z}|\,\mathrm{tail}(\mathtt{P})) \times \mathtt{P.b} & \mathtt{Pz} \notin T \text{ and } \mathtt{P} \in T \\ \mathtt{Pz.p} & \mathtt{Pz} \in T \end{cases} \tag{4}$$

The optimistic version (or "max-backoff") q of p is defined as $q(\mathtt{z}|\mathtt{P}) \equiv \max_H p(\mathtt{z}|\mathtt{HP})$, where $\mathtt{H}$ varies over all possible contexts extending the prefix $\mathtt{P}$ to the left. The Max-ARPA table allows to compute $q(\mathtt{z}|\mathtt{P})$ for arbitrary values of $\mathtt{z}$ and $\mathtt{P}$. It is constructed on the basis of the ARPA table T by adding two columns to T: a column $\mathtt{Z.q}$ that stores the value $q(\mathtt{z}|\mathtt{P})$ and a column $\mathtt{Z.m}$ that stores an optimistic version of the backoff weight.

These columns are computed offline in two passes by first sorting T in descending order of n-gram length.[12] In the first pass (Algorithm 1), we compute for every entry in the table an optimistic backoff weight $\mathtt{m}$. In the second pass (Algorithm 2), we compute for every entry an optimistic conditional probability $\mathtt{q}$ by maximising over 1-word history extensions (whose $\mathtt{.q}$ fields are already known due to the sorting of T).

The following **Theorem** holds (see proof below): *For an arbitrary n-gram $Z = Pz$, the probability $q(z|P)$ can be recovered through the procedure shown in Equation 5.*

$$q(\mathtt{z}|\mathtt{P}) = \begin{cases} p(\mathtt{z}|\mathtt{P}) & \mathtt{Pz} \notin T \text{ and } \mathtt{P} \notin T \\ p(\mathtt{z}|\mathtt{P}) \times \mathtt{P.m} & \mathtt{Pz} \notin T \text{ and } \mathtt{P} \in T \\ \mathtt{Pz.q} & \mathtt{Pz} \in T \end{cases} \tag{5}$$

Note that, if $\mathtt{Z}$ is listed in the table, we return its upperbound probability $\mathtt{q}$ directly. When the n-gram is unknown, but its prefix is known, we take into account the optimistic backoff weight $\mathtt{m}$ of the prefix. On the other hand, if both the n-gram and its prefix are unknown, then no additional context could change the score of the n-gram, in which case $q(\mathtt{z}|\mathtt{P}) = p(\mathtt{z}|\mathtt{P})$.

In the sequel, we will need the following definitions. Suppose $\alpha = y_I^J$ is a substring of $\mathbf{y} = y_1^M$. Then $p_{\mathrm{LM}}(\alpha) \equiv \prod_{k=I}^{J} p(y_k|y_1^{k-1})$ is the contribution of α to the true LM score of $\mathbf{y}$. We then obtain an upperbound $q_{\mathrm{LM}}(\alpha)$ to this contribution by defining $q_{\mathrm{LM}}(\alpha) \equiv q(y_I|\epsilon) \prod_{k=I+1}^{J} q(y_k|y_I^{k-1})$.

[12] If an n-gram is listed in T, then all its substrings must also be listed. Certain pruning strategies may corrupt this property, in which case we make missing substrings explicit.

Proof of **Theorem**. Let us first suppose that the length of $\mathtt{P}$ is strictly larger than the order n of the language model. Then for any $\mathtt{H}$, $p(\mathtt{z}|\mathtt{HP}) = p(\mathtt{z}|\mathtt{P})$; this is because $\mathtt{HP} \notin T$ and $\mathtt{P} \notin T$, along with all intermediary strings, hence, by (4), $p(\mathtt{z}|\mathtt{HP}) = p(\mathtt{z}|\,\mathrm{tail}(\mathtt{HP})) = p(\mathtt{z}|\,\mathrm{tail}(\mathrm{tail}(\mathtt{HP}))) = \ldots = p(\mathtt{z}|\mathtt{P})$. Hence $q(\mathtt{z}|\mathtt{P}) = p(\mathtt{z}|\mathtt{P})$, and, because $\mathtt{Pz} \notin T$ and $\mathtt{P} \notin T$, the theorem is satisfied in this case.

Having established the theorem for $|\mathtt{P}| > n$, we now assume that it is true for $|\mathtt{P}| > m$ and prove by induction that it is true for $|\mathtt{P}| = m$. We use the fact that, by the definition of $\mathtt{q}$, we have $q(\mathtt{z}|\mathtt{P}) = \max_{\mathtt{x}\in\Delta} q(\mathtt{z}|\mathtt{xP})$. We have three cases to consider.

First, suppose that $\mathtt{Pz} \notin T$ and $\mathtt{P} \notin T$. Then $\mathtt{xPz} \notin T$ and $\mathtt{xP} \notin T$, hence by induction $q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P})$ for any $\mathtt{x}$, therefore $q(\mathtt{z}|\mathtt{P}) = p(\mathtt{z}|\mathtt{P})$. We have thus proven the first case.

Second, suppose that $\mathtt{Pz} \notin T$ and $\mathtt{P} \in T$. Then, for any $\mathtt{x}$, we have $\mathtt{xPz} \notin T$, and:

$$\begin{aligned} q(\mathtt{z}|\mathtt{P}) &= \max_{\mathtt{x}\in\Delta} q(\mathtt{z}|\mathtt{xP}) \\ &= \max\Big(\max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\notin T} q(\mathtt{z}|\mathtt{xP}), \max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\in T} q(\mathtt{z}|\mathtt{xP})\Big). \end{aligned}$$

For $\mathtt{xP} \notin T$, by induction, $q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P})$, and therefore $\max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\notin T} q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P})$. For $\mathtt{xP} \in T$, we have $q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{xP}) \times \mathtt{xP.m} = p(\mathtt{z}|\mathtt{P}) \times \mathtt{xP.b} \times \mathtt{xP.m}$. Thus, we have:

$$\max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\in T} q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P}) \times \max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\in T} \mathtt{xP.b} \times \mathtt{xP.m}.$$

But now, because of lines 3 and 4 of Algorithm 1, $\mathtt{P.m} = \max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\in T} \mathtt{xP.b} \times \mathtt{xP.m}$, hence $\max_{\mathtt{x}\in\Delta,\, \mathtt{xP}\in T} q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P}) \times \mathtt{P.m}$. Therefore, $q(\mathtt{z}|\mathtt{P}) = \max(p(\mathtt{z}|\mathtt{P}), p(\mathtt{z}|\mathtt{P}) \times \mathtt{P.m}) = p(\mathtt{z}|\mathtt{P}) \times \mathtt{P.m}$, where we have used the fact that $\mathtt{P.m} \geq 1$ due to line 1 of Algorithm 1. We have thus proven the second case.

Finally, suppose that $\mathtt{Pz} \in T$. Then, again,

$$\begin{aligned} q(\mathtt{z}|\mathtt{P}) &= \max_{\mathtt{x}\in\Delta} q(\mathtt{z}|\mathtt{xP}) \\ &= \max\Big(\\ &\qquad \max_{\mathtt{x}\in\Delta,\, \mathtt{xPz}\notin T,\, \mathtt{xP}\notin T} q(\mathtt{z}|\mathtt{xP}), \\ &\qquad \max_{\mathtt{x}\in\Delta,\, \mathtt{xPz}\notin T,\, \mathtt{xP}\in T} q(\mathtt{z}|\mathtt{xP}), \\ &\qquad \max_{\mathtt{x}\in\Delta,\, \mathtt{xPz}\in T} q(\mathtt{z}|\mathtt{xP})\ \Big). \end{aligned}$$

For $\mathtt{xPz} \notin T$, $\mathtt{xP} \notin T$, we have $q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{P}) = \mathtt{Pz.p}$, where the last equality is due to the fact that $\mathtt{Pz} \in T$. For $\mathtt{xPz} \notin T$, $\mathtt{xP} \in T$, we have $q(\mathtt{z}|\mathtt{xP}) = p(\mathtt{z}|\mathtt{xP}) \times \mathtt{xP.m} = p(\mathtt{z}|\mathtt{P}) \times \mathtt{xP.b} \times \mathtt{xP.m} = \mathtt{Pz.p} \times \mathtt{xP.b} \times \mathtt{xP.m}$. For $\mathtt{xPz} \in T$, we have $q(\mathtt{z}|\mathtt{xP}) = \mathtt{xPz.q}$. Overall, we thus have:

$$\begin{aligned} q(\mathtt{z}|\mathtt{P}) = \max\Big(& \mathtt{Pz.p}, \\ & \max_{\mathtt{x}\in\Delta,\, \mathtt{xPz}\notin T,\, \mathtt{xP}\in T} \mathtt{Pz.p} \times \mathtt{xP.b} \times \mathtt{xP.m}, \\ & \max_{\mathtt{x}\in\Delta,\, \mathtt{xPz}\in T} \mathtt{xPz.q}\ \Big). \end{aligned}$$

Note that $\mathtt{xPz} \in T \Rightarrow \mathtt{xP} \in T$, and then one can check that Algorithm 2 exactly computes $\mathtt{Pz.q}$ as this maximum over three maxima, hence $\mathtt{Pz.q} = q(\mathtt{z}|\mathtt{P})$.

Algorithm 1 Max-ARPA: first pass

```
1: for Z ∈ T do
2:     Z.m ← 1
3:     for x ∈ Δ s.t xZ ∈ T do
4:         Z.m ← max(Z.m, xZ.b × xZ.m)
5:     end for
6: end for
```

Algorithm 2 Max-ARPA: second pass

```
1: for Z = Pz ∈ T do
2:     Pz.q ← Pz.p
3:     for x ∈ Δ s.t xP ∈ T do
4:         if xPz ∈ T then
5:             Pz.q ← max(Pz.q, xPz.q)
6:         else
7:             Pz.q ← max(Pz.q, Pz.p × xP.b × xP.m)
8:         end if
9:     end for
10: end for
```

4.4 Search

The search for the true optimum derivation is illustrated in Algorithm 3. The algorithm takes as input the initial proposal distribution $g^{(0)}(\mathbf{d})$ (see §4.2, Figure 3) and a maximum error ϵ (which we set to a small constant 0.001 rather than zero, to avoid problems with floating point precision). In line 3 we find the optimum derivation $\mathbf{d}$ in $g^{(0)}$ (see §4.5). The variable g^* stores the maximum score w.r.t. the current proposal, while the variable f^* stores the maximum score observed thus far w.r.t. the true model (note that in line 5 we assess the true score of $\mathbf{d}$). In line 6 we start a loop that runs until the error falls below ϵ. This *error* is the difference (in $\log$-domain) between the proxy maximum g^* and the best true score observed thus far f^*.[13] In line 7, we refine the current proposal using evidence from $\mathbf{d}$ (see §4.6). In line 9, we update the maximum derivation searching through the refined proposal. In line 11, we keep track of the best score so far according to the true model, in order to compute the updated gap in line 6.

4.5 Dynamic Programming

Finding the best derivation in a proposal hypergraph is straightforward with standard dynamic programming. We can compute *inside weights* in the *max-times* semiring in time proportional to $O(|V| + |E|)$ (Goodman, 1999). Once inside weights have been computed, finding the Viterbi-derivation starting from the root is straightforward. A simple, though important, optimisation concerns the computation of inside weights. The inside algorithm (Baker, 1979) requires a bottom-up traverse of the nodes in V. To do that, we topologically sort the nodes in V at time $t = 0$ and maintain a sorted list of nodes as we refine g throughout the search – thus avoiding having to recompute the partial ordering of the nodes at every iteration.

[13] Because $g^{(t)}$ upperbounds f everywhere, in optimisation we have a guarantee that the maximum of f must lie in the interval $[f^*, g^*)$ (see Figure 2) and the quantity $g^* - f^*$ is an upperbound on the error that we incur if we early-stop the search at any given time t. This bound provides a principled criterion in trading accuracy for performance (a direction that we leave for future work). Note that most algorithms for approximate search produce solutions with unbounded error.

Algorithm 3 Exact decoding

```
1: function OPTIMISE(g^(0), ε)
2:     t ← 0                                          ▷ step
3:     d ← argmax_d g^(t)(d)
4:     g* ← g^(t)(d)
5:     f* ← f(d)
6:     while (g* − f* ≥ ε) do      ▷ ε is the maximum error
7:         g^(t+1) ← refine(g^(t), d)        ▷ update proposal
8:         t ← t + 1
9:         d ← argmax_d g^(t)(d)             ▷ update argmax
10:        g* ← g^(t)(d)
11:        f* ← max(f*, f(d))          ▷ update "best so far"
12:    end while
13:    return g^(t), d
14: end function
```

4.6 Refinement

If a derivation $\mathbf{d} = \operatorname{argmax}_{\mathbf{d}} g^{(t)}(\mathbf{d})$ is such that $g^{(t)}(\mathbf{d}) \ll f(\mathbf{d})$, there must be in $\mathbf{d}$ at least one n-gram whose upperbound LM weight is far above its true LM weight. We then lower $g^{(t)}$ locally by refining only nonterminal nodes that participate in $\mathbf{d}$. Nonterminal nodes are refined by having their LM states extended one word at a time.[14]

For an illustration, assume we are performing optimisation with a bigram LM. Suppose that in the first iteration a derivation $\mathbf{d}_0 = \operatorname{argmax}_{\mathbf{d}} g^{(0)}(\mathbf{d})$ is obtained. Now consider an edge in $\mathbf{d}_0$

$$[l, C, r, \epsilon]\ \ \alpha y_1 \xrightarrow{w} [l_0, C_0, r_0, \epsilon]$$

where an empty LM state is made explicit (with an empty string ϵ) and αy_1 represents a target phrase. We refine the edge's head $[l_0, C_0, r_0, \epsilon]$ by creating a node based on it, however, with an extended LM state, i.e., $[l_0, C_0, r_0, y_1]$. This motivates a split of the set of incoming edges to the original node, such that, if the target projection of an incoming

[14] The refinement operation is a special case of a general finite-state intersection. However, keeping its effect local to derivations going through a specific node is non-trivial using the general mechanism and justifies a tailored operation.

edge ends in y_1, that edge is reconnected to the new node as below.

$$[l, C, r, \epsilon]\ \ \alpha y_1 \xrightarrow{w} [l_0, C_0, r_0, y_1]$$

The outgoing edges from the new node are reweighted copies of those leaving the original node. That is, outgoing edges such as

$$[l_0, C_0, r_0, \epsilon]\ \ y_2\beta \xrightarrow{w} \left[l', C', r', \gamma'\right]$$

motivate edges such as

$$[l_0, C_0, r_0, y_1]\ \ y_2\beta \xrightarrow{w \otimes w'} \left[l', C', r', \gamma'\right]$$

where $w' = \lambda_\psi q_{\text{LM}}(y_1 y_2) / q_{\text{LM}}(y_2)$ is a change in LM probability due to an extended context.

Figure 4 is the logic program that constructs the refined hypergraph in the general case. In comparison to Figure 3, items are now extended to store an LM state. The input is the original hypergraph $G = \langle V, E \rangle$ and a node $\mathbf{v}_0 \in V$ to be refined by left-extending its LM state γ_0 with the word y. In the program, $\left\langle \mathbf{u}\sigma \xrightarrow{w} \mathbf{v} \right\rangle$ with $\mathbf{u}, \mathbf{v} \in V$ and $\sigma \in \Delta^*$ represents an edge in E. An item $[l, C, r, \gamma]_{\mathbf{v}}$ (annotated with a state $\mathbf{v} \in V$) represents a node (in the refined hypergraph) whose signature is equivalent to $\mathbf{v}$ (in the input hypergraph). We start with AXIOMS by copying the nodes in G. In COPY, edges from G are copied unless they are headed by $\mathbf{v}_0$ and their target projections end in $y\gamma_0$ (the extended context). Such edges are processed by REFINE, which instead of copying them, creates new ones headed by a refined version of $\mathbf{v}_0$. Finally, REWEIGHT continues from the refined node with reweighted copies of the edges leaving $\mathbf{v}_0$. The weight update represents a change in LM probability (w.r.t. the upper-bound distribution) due to an extended context.

INPUT
$G = \langle V, E \rangle$
$\mathbf{v}_0 = [l_0, C_0, r_0, \gamma_0] \in V$ where $\gamma_0 \in \Delta^*$
$y \in \Delta$
ITEM $[l, C, r, \gamma \in \Delta^*]$
AXIOMS

$$\frac{}{[l, C, r, \gamma]_{\mathbf{v}}}\ \ \mathbf{v} \in V$$

COPY

$$\frac{[l, C, r, \alpha]_{\mathbf{u}} \quad \left\langle \mathbf{u}\beta \xrightarrow{w} \mathbf{v} \right\rangle}{[l', C', r', \alpha']_{\mathbf{v}} : w} \quad \begin{array}{l} \mathbf{v} \neq \mathbf{v}_0 \vee \alpha\beta \neq \sigma y \gamma_0 \\ \alpha, \alpha', \beta, \sigma \in \Delta^* \end{array}$$

REFINE

$$\frac{[l, C, R, \alpha]_{\mathbf{u}} \quad \left\langle \mathbf{u}\beta \xrightarrow{w} \mathbf{v}_0 \right\rangle}{[l_0, C_0, r_0, y\gamma_0] : w} \quad \begin{array}{l} \alpha\beta = \sigma y \gamma_0 \\ \alpha, \beta, \sigma \in \Delta^* \end{array}$$

REWEIGHT

$$\frac{[l_0, C_0, r_0, y\gamma_0] \quad \left\langle \mathbf{v}_0\sigma \xrightarrow{w} \mathbf{v} \right\rangle}{[l, C, r, \gamma]_{\mathbf{v}} : w \otimes w'}\ \ \sigma, \gamma \in \Delta^*$$

where $w' = \lambda_\psi \dfrac{q_{\text{LM}}(y\gamma_0)}{q_{\text{LM}}(\gamma_0)}$

Figure 4: Local intersection via LM right state refinement. The input is a hypergraph $G = \langle V, E \rangle$, a node $\mathbf{v}_0 \in V$ singly identified by its carry $[l_0, C_0, r_0, \gamma_0]$ and a left-extension y for its LM context γ_0. The program copies most of the edges $\left\langle \mathbf{u}\sigma \xrightarrow{w} \mathbf{v} \right\rangle \in E$. If a derivation goes through $\mathbf{v}_0$ and the string under $\mathbf{v}_0$ ends in $y\gamma_0$, the program refines and reweights it.

5 Experiments

We used the dataset made available by the Workshop on Statistical Machine Translation (WMT) (Bojar et al., 2013) to train a German-English phrase-based system using the Moses toolkit (Koehn et al., 2007) in a standard setup. For phrase extraction, we used both Europarl (Koehn, 2005) and News Commentaries (NC) totalling about 2.2M sentences.[15] For language modelling, in addition to the monolingual parts of Europarl and NC, we added News-2013 totalling about 25M sentences. We performed language model interpolation and *batch-mira* tuning (Cherry and Foster, 2012) using *newstest2010* (2,849 sentence pairs). For tuning we used cube pruning with a large beam size ($k = 5000$) and a distortion limit $d = 4$. Unpruned language models were trained using *lmplz* (Heafield et al., 2013b) which employs modified Kneser-Ney smoothing (Kneser and Ney, 1995). We report results on *newstest2012*.

Our exact decoder produces optimal translation derivations for all the 3,003 sentences in the test set. Table 1 summarises the performance of our novel decoder for language models of order $n = 3$ to $n = 5$. For 3-gram LMs we also varied the distortion limit d (from 4 to 6). We report the average time (in seconds) to build the initial proposal, the total run time of the algorithm, the number of iterations N before convergence, and the size of the hypergraph in the end of the search (in thousands of nodes and thousands of edges).[16]

[15] Pre-processing: tokenisation, truecasing and automatic compound-splitting (German only). Following Durrani et al. (2013), we set the maximum phrase length to 5.

[16] The size of the initial proposal does not depend on LM order, but rather on distortion limit (see Figure 3): on average (in thousands) $|V_0| = 0.6$ and $|E_0| = 27$ with $d = 4$, $|V_0| = 1.3$ and $|E_0| = 70$ with $d = 5$, and $|V_0| = 2.5$ and

n	d	build (s)	total (s)	N	$\|V\|$	$\|E\|$
3	4	1.5	21	190	2.5	159
3	5	3.5	55	303	4.4	343
3	6	10	162	484	8	725
4	4	1.5	50	350	4	288
5	4	1.5	106	555	6.1	450

Table 1: Performance of the exact decoder in terms of: time to *build* $g^{(0)}$, *total* decoding time including *build*, number of *iterations* (N), and number of nodes and edges (in thousands) at the end of the search.

n	Nodes at level m					LM states at level m			
	0	1	2	3	4	1	2	3	4
3	0.4	1.2	0.5	-	-	113	263	-	-
4	0.4	1.6	1.4	0.3	-	132	544	212	-
5	0.4	2.1	2.4	0.7	0.1	142	790	479	103

Table 2: Average number of nodes (in thousands) whose LM state encode an m-gram, and average number of unique LM states of order m in the final hypergraph for different n-gram LMs ($d = 4$ everywhere).

It is insightful to understand how different aspects of the initial proposal impact on performance. Increasing the translation option limit (*tol*) leads to $g^{(0)}$ having more edges (this dependency is linear with *tol*). In this case, the number of nodes is only minimally affected — due to the possibility of a few new segmentations. The maximum phrase length (*mpl*) introduces in $g^{(0)}$ more configurations of reordering constraints ($[l, C]$ in Figure 3). However, not many more, due to C being limited by the distortion limit d. In practice, we observe little impact on time performance. Increasing d introduces many more permutations of the input leading to exponentially many more nodes and edges. Increasing the order n of the LM has no impact on $g^{(0)}$ and its impact on the overall search is expressed in terms of a higher number of nodes being locally intersected.

An increased hypergraph, be it due to additional nodes or additional edges, necessarily leads to slower iterations because at each iteration we must compute inside weights in time $O(|V|+|E|)$. The number of nodes has the larger impact on the number of iterations. OS* is very efficient in ignoring hypotheses (edges) that cannot compete for an optimum. For instance, we observe that running time depends linearly on *tol* only through the computation of inside weights, while the number of iterations is only minimally affected.[17] An increased LM order, for a fixed distortion limit, impacts much more on the number of iterations than on the average running time of a single iteration. Fixing $d = 4$, the average time per iteration is 0.1 ($n = 3$), 0.13 ($n = 4$) and 0.18 ($n = 5$). Fixing a 3-gram LM, we observe 0.1 ($d = 4$), 0.17 ($d = 5$) and 0.31 ($d = 6$). Note the exponential growth of the latter, due to a proposal encoding exponentially many more permutations.

Table 2 shows the average degree of refinement of the nodes in the final proposal. Nodes are shown by level of refinement, where m indicates that they store m words in their carry. The table also shows the number of unique m-grams ever incorporated to the proposal. This table illustrates well how our decoding algorithm moves from a coarse upperbound where every node stores an empty string to a variable-order representation which is sufficient to prove an optimum derivation.

In our approach a complete derivation is optimised from the proxy model at each iteration. We observe that over 99% of these derivations project onto distinct strings. In addition, while the optimum solution may be found early in the search, a certificate of optimality requires refining the proxy until convergence (see §4.1). It turns out that most of the solutions are first encountered as late as in the last 6-10% of the iterations.

We use the optimum derivations obtained with our exact decoder to measure the number of search errors made by beam search and cube pruning with increasing beam sizes (see Table 3). Beam search reaches optimum derivations with beam sizes $k \geq 500$ for all language models tested. Cube pruning, on the other hand, still makes mistakes at $k = 1000$. Table 4 shows translation quality achieved with different beam sizes for cube pruning and compares it to exact decoding. Note that for $k \geq 10^4$ cube pruning converges to optimum

$|E_0| = 178$ with $d = 6$. Observe the exponential dependency on distortion limit, which also leads to exponentially longer running times.

[17] It is possible to reduce the size of the hypergraph throughout the search using the upperbound on the search error $g^* - f^*$ to prune hypotheses that surely do not stand a chance of competing for the optimum (Graehl, 2005). Another direction is to group edges connecting the same nonterminal nodes into one *partial edge* (Heafield et al., 2013a) — this is particularly convenient due to our method only visiting the 1-best derivation from $g(\mathbf{d})$ at each iteration.

k	Beam search			Cube pruning		
	3	4	5	3	4	5
10	938	1294	1475	2168	2347	2377
10^2	19	60	112	613	999	1126
10^3	0	0	0	29	102	167
10^4	0	0	0	0	4	7

Table 3: Beam search and cube pruning search errors (out of 3,003 test samples) by beam size using LMs of order 3 to 5 ($d = 4$).

order	3			4	5
k	$d = 4$	$d = 5$	$d = 6$	$d = 4$	$d = 4$
10	20.47	20.13	19.97	20.71	20.69
10^2	21.14	21.18	21.08	21.73	21.76
10^3	21.27	21.34	21.32	21.89	21.91
10^4	21.29	21.37	21.37	21.92	21.93
OS*	21.29	21.37	21.37	21.92	21.93

Table 4: Translation quality in terms of BLEU as a function of beam size in cube pruning with language models of order 3 to 5. The bottom row shows BLEU for our exact decoder.

derivations in the vast majority of the cases (100% with a 3-gram LM) and translation quality in terms of BLEU is no different from OS*. However, with $k < 10^4$ both model scores and translation quality can be improved. Figure 5 shows a finer view on search errors as a function of beam size for LMs of order 3 to 5 (fixed $d = 4$). In Figure 6, we fix a 3-gram LM and vary the distortion limit (from 4 to 6). Dotted lines correspond to beam search and dashed lines correspond to cube pruning.

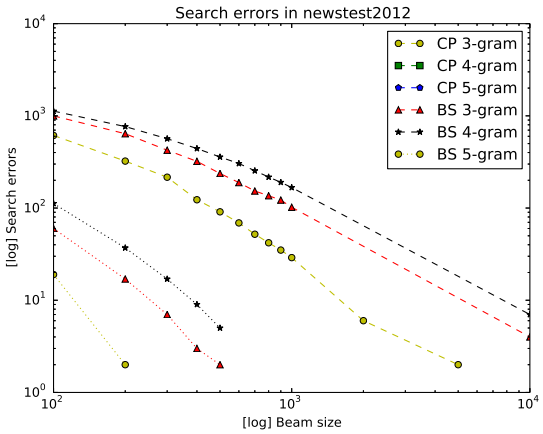

Figure 5: Search errors made by beam search and cube pruning as a function of beam-size.

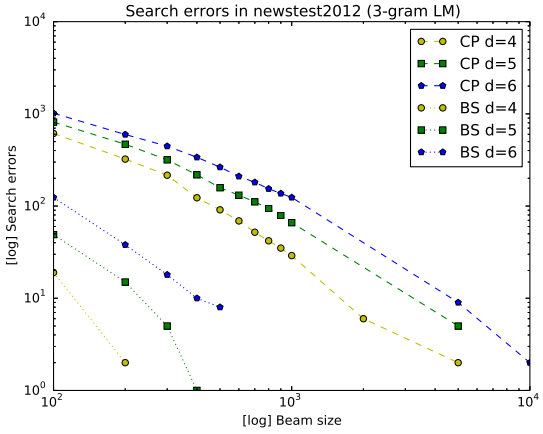

Figure 6: Search errors made by beam search and cube pruning as a function of the distortion limit (decoding with a 3-gram LM).

6 Conclusions and Future Work

We have presented an approach to decoding with unpruned hypergraphs using upperbounds on the language model distribution. The algorithm is an instance of a coarse-to-fine strategy with connections to A* and adaptive rejection sampling known as OS*. We have tested our search algorithm using state-of-the-art phrase-based models employing robust language models. Our algorithm is able to decode all sentences of a standard test set in manageable time consuming very little memory. We have performed an analysis of search errors made by beam search and cube pruning and found that both algorithms perform remarkably well for phrase-based decoding. In the case of cube pruning, we show that model score and translation quality can be improved for beams $k < 10,000$.

There are a number of directions that we intend to investigate to speed up our decoder, such as: (1) error-safe pruning based on search error bounds; (2) use of reinforcement learning to guide the decoder in choosing which n-gram contexts to extend; and (3) grouping edges into partial edges, effectively reducing the size of the hypergraph and ultimately computing inside weights in less time.

Acknowledgments

The work of Wilker Aziz and Lucia Specia was supported by EPSRC (grant EP/K024272/1).

References

Michael Auli, Adam Lopez, Hieu Hoang, and Philipp Koehn. 2009. A systematic analysis of translation model search spaces. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, StatMT '09, pages 224–232, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wilker Aziz, Marc Dymetman, and Sriram Venkatapathy. 2013. Investigations in exact inference for hierarchical translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 472–483, Sofia, Bulgaria, August. Association for Computational Linguistics.

James K. Baker. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA, June.

Ondřej Bojar, Christian Buck, Chris Callison-Burch, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2013. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria, August. Association for Computational Linguistics.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, June.

Simon Carter, Marc Dymetman, and Guillaume Bouchard. 2012. Exact sampling and decoding in high-order hidden Markov models. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1125–1134, Jeju Island, Korea, July. Association for Computational Linguistics.

Yin-Wen Chang and Michael Collins. 2011. Exact decoding of phrase-based translation models through Lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 26–37, Stroudsburg, PA, USA. Association for Computational Linguistics.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 427–436, Stroudsburg, PA, USA. Association for Computational Linguistics.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Shay B. Cohen, Robert J. Simmons, and Noah A. Smith. 2008. Dynamic programming algorithms as products of weighted logic programs. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg.

G Dantzig, R Fulkerson, and S Johnson. 1954. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410.

Nadir Durrani, Barry Haddow, Kenneth Heafield, and Philipp Koehn. 2013. Edinburgh's machine translation systems for European language pairs. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 114–121, Sofia, Bulgaria, August. Association for Computational Linguistics.

Chris Dyer and Philip Resnik. 2010. Context-free reordering, finite-state translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 858–866, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marc Dymetman, Guillaume Bouchard, and Simon Carter. 2012. Optimization and sampling for NLP from a unified viewpoint. In *Proceedings of the First International Workshop on Optimization Techniques for Human Language Technology*, pages 79–94, Mumbai, India, December. The COLING 2012 Organizing Committee.

Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, April.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 228–235, Stroudsburg, PA, USA. Association for Computational Linguistics.

Joshua Goodman. 1999. Semiring parsing. *Comput. Linguist.*, 25(4):573–605, December.

Jonathan Graehl. 2005. Relatively useless pruning. Technical report, USC Information Sciences Institute.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions On Systems Science And Cybernetics*, 4(2):100–107.

Kenneth Heafield, Philipp Koehn, and Alon Lavie. 2013a. Grouping language model boundary words

to speed k-best extraction from hypergraphs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 958–968, Atlanta, Georgia, USA, June.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013b. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria, August. Association for Computational Linguistics.

Kenneth Heafield, Michael Kayser, and Christopher D. Manning. 2014. Faster Phrase-Based decoding by refining feature state. In *Proceedings of the Association for Computational Linguistics*, Baltimore, MD, USA, June.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June. Association for Computational Linguistics.

Gonzalo Iglesias, Adrià de Gispert, Eduardo R. Banga, and William Byrne. 2009. Rule filtering by pattern for efficient hierarchical translation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 380–388, Athens, Greece, March. Association for Computational Linguistics.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Series in Artificial Intelligence. Prentice Hall, 1 edition.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. *Acoustics, Speech, and Signal Processing*, 1:181–184.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Comput. Linguist.*, 25(4):607–615, December.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit*, pages 79–86.

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

Zhifei Li and Sanjeev Khudanpur. 2008. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In *Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, SSST '08, pages 10–18, Stroudsburg, PA, USA. Association for Computational Linguistics.

Adam Lopez. 2008. Statistical machine translation. *ACM Computing Surveys*, 40(3):8:1–8:49, August.

Adam Lopez. 2009. Translation as weighted deduction. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 532–540, Stroudsburg, PA, USA. Association for Computational Linguistics.

Franz Josef Och, Nicola Ueffing, and Hermann Ney. 2001. An efficient A* search algorithm for statistical machine translation. In *Proceedings of the workshop on Data-driven methods in machine translation - Volume 14*, DMMT '01, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, volume 1 of *ACL '03*, pages 160–167, Stroudsburg, PA, USA. Association for Computational Linguistics.

Slav Petrov, Aria Haghighi, and Dan Klein. 2008. Coarse-to-fine syntactic machine translation using language projections. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 108–116, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sebastian Riedel and James Clarke. 2009. Revisiting optimal decoding for machine translation IBM model 4. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, pages 5–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christian P. Robert and George Casella. 2004. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through Lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 72–82, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christoph Tillmann, Stephan Vogel, Hermann Ney, and A. Zubiaga. 1997. A DP based search using monotone alignments in statistical translation. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 289–296, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mikhail Zaslavskiy, Marc Dymetman, and Nicola Cancedda. 2009. Phrase-based statistical machine translation as a traveling salesman problem. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 333–341, Stroudsburg, PA, USA. Association for Computational Linguistics.