Noname manuscript No.
(will be inserted by the editor)

Cardinality Pruning and Language Model Heuristics for Hierarchical Phrase-Based Translation

David Vilar · **Hermann Ney**

Received: date / Accepted: date

Abstract In this article we present two novel enhancements for the cube pruning and cube growing algorithms, two of the most widely applied methods when using the hierarchical approach to statistical machine translation.

Cube pruning is the de-facto standard search algorithm for the hierarchical model. We propose to adapt concepts of the source cardinality synchronous search organization as used for standard phrase-based translation to the characteristics of cube pruning. In this way we will be able to improve the performance of the generation process and reduce the average translation time per sentence to approximately one fourth.

We will also investigate the cube growing algorithm, a reformulation of cube pruning with on-demand computation. This algorithm depends on a heuristic for the language model, but this issue is barely discussed in the original work. We analyze the behaviour of this heuristic and propose a new one which greatly reduces memory consumption without costs in running time or translation performance. Results are reported on the German-English Europarl corpus.

Keywords Hierarchical phrase-based model · Search algorithm · Efficient search · Cube pruning · Cube growing

1 Introduction

The hierarchical phrase-based model, first presented by Chiang (2005) has proved to be a very effective approach for statistical machine translation. It can be considered a generalization of the standard phrase-based approach, where the phrases are allowed to have gaps. These gaps are then filled with other hierarchical phrases to produce a complete translation. The model is formally defined as a synchronous context-free grammar. As such the translation process can be considered as a generalization of a parsing procedure.

D. Vilar, H. Ney
RWTH Aachen University
Ahornstrasse 55
D-52056 Aachen (Germany)
E-mail: {vilar,ney}@cs.rwth-aachen.de
Present address of D. Vilar: DFKI GmbH, Language Technoloy Lab, Germany, E-mail: david.vilar@dfki.de

The two most widely used algorithms for hierarchical models are the cube pruning and cube growing algorithms. In this article we present two novel extensions for these algorithms:

- For the cube pruning algorithm, we will analyze its behaviour in terms of translation quality depending on computational effort and address one of its deficiencies, the overgeneration of derivations that are not needed for producing the final translation. We will adapt the principle of source cardinality synchronous search widely used in standard phrase-based translation to the hierarchical model and improve the computational efficiency of the search process reducing running time to nearly one fourth, without compromising translation quality.
- The cube growing algorithm is a reformulation of the cube pruning algorithm, with an on-demand computation strategy (Huang and Chiang, 2007). The performance of this algorithm critically depends on a heuristic for the language model costs, however this issue is hardly discussed in the original paper. In this article we analyze the performance of the heuristic suggested by Huang and Chiang (2007) and propose a new one based on word-clustering techniques. With this new heuristic we are able to achieve the same performance as with the original heuristic with a fraction of the memory consumption.

This article is organized as follows: in Section 2 we will give an overview of related work. In Section 3 we will discuss some preliminaries and introduce the notation we will use in the rest of the paper. Section 4 will formalize the hierarchical phrase-base model and the search problem. Some issues that have to be dealt with when adapting a parsing algorithm for the hierarchical task will be discussed in Section 5. In Section 6 and 7, we will analyze and expand the cube pruning and cube growing algorithms, respectively. Experimental results comparing the results of all the presented algorithms are discussed in Section 8. The article concludes in Section 9.

2 Related Work

The first time the concept of hierarchical phrases appears in the literature of machine translation, albeit with the name *pattern pairs*, is in the paper by Block (2000) on example based machine translation. Zens (2002) shortly discusses this concept and suggests applying it to statistical machine translation. However, this line of work was not pursued further.

The current reference work for the hierarchical phrase-based translation approach is (Chiang, 2005) and its extension as a journal paper (Chiang, 2007). A detailed analysis of the search problem as a deductive system can be found in (Lopez, 2009) and Hopkins and Langmead (2009) present the cube pruning algorithm as an instantiation of A* with an appropriate search space. In this article we will take an different approach and formulate the search as a dynamic programming problem.

As will be discussed later, the translation process in a hierarchical phrase-based system can be thought of as two parts that are more or less independent of each other: the parsing procedure, and the computation of language model scores on the implicitly generated translations.

Parsing is one of the classical topics in computer science. Therefore a vast set of literature dealing with the problem exists. One of the most widely used algorithms is the CYK algorithm, named after the three authors that developed the algorithm independently: Cocke (1969), Younger (1967) and Kasami (1965). In this article we will apply the so-called CYK+ algorithm, a generalisation of CYK by Chappelier and Rajman (1998).

The inclusion of language model information in the translation process is also described by Chiang (2007), where the cube pruning algorithm is introduced. A more detailed explanation can be found in (Huang and Chiang, 2007), where the cube growing algorithm is introduced, an on-demand version of the cube pruning algorithm. Both algorithms are adaptations of the n-best derivations algorithms for context free grammars introduced in (Huang and Chiang, 2005), extended for dealing with the peculiarities of statistical machine translation. Cube pruning can be considered nowadays to be the standard generation procedure for the hierarchical model. It is implemented in most decoders, including the freely available Joshua (Li et al, 2009a), SAMT (Zollmann and Venugopal, 2006) and Moses (Koehn et al, 2007).

For both algorithms, cube pruning and cube growing, we will present an extension to improve their performance. The first one will adapt concepts of the search process of the standard phrase-based translation approach to the hierarchical phrase-based translation. A good overview of these concepts can be found in (Zens, 2008). The second extension will deal with computing a heuristic for the language model computation and is similar in spirit to (Petrov et al, 2008).

There are other approaches for dealing with the generation process in the hierarchical phrase-based approach. One of the first alternative approaches was presented by Watanabe et al (2006), where the authors adapt the standard left-to-right generation approach widely used in standard phrase-based translation to a restricted subset of the hierarchical phrase-based model. This approach can be generalized to the unconstrained hierarchical model, but then the search process is less efficient as when using the standard algorithms.

Another alternative is the two pass approach based on n-best lists presented in (Venugopal et al, 2007) and implemented in the first versions of the open source decoder SAMT[1] (Zollmann and Venugopal, 2006). One further possibility is to formulate the translation model as a tree automaton and apply some of the standard methods for dealing with such models (May and Knight, 2006).

Another recent approach represents the space of possible translations of a given source sentence as a finite state automaton and computes the language model score via finite state automata composition (Iglesias et al, 2009). This approach allows for easy inclusion of other techniques like forward-backward pruning or Bayes risk decoding.

3 Preliminaries

In this section we will shortly discuss some concepts that will be necessary for the following exposition and will aid in presenting the notation we will use throughout the article.

3.1 Derivations

An important concept for developing the following algorithms will be the concept of *derivation*. Given a monolingual context-free grammar (CFG), a derivation represents a string transformation process where each string is obtained from the previous one by applying a rule from the grammar. More formally, given the grammar $G = (N, \Sigma, R, S)$, two strings $\alpha, \gamma \in (\Sigma \cup N)^\star$ and a rule $r = A \to \beta \in R$, we can apply the rule r to the string $\alpha A \gamma$, obtaining the string $\alpha\beta\gamma$. We use the notation $\alpha A \gamma \Rightarrow \alpha\beta\gamma$. We define $\overset{*}{\Rightarrow}$ as the reflexive and

[1] Newer versions do not use this approach anymore and instead implement cube pruning.

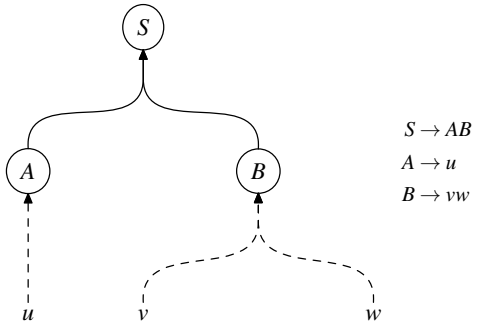


Fig. 1 Example of a hypergraph corresponding to a grammar derivation. The hypergraph corresponds to the derivation of the string *uvw* using the grammar shown on the right. The dashed hyperedges do not strictly correspond to the hypergraph, but are included for clarity.

transitive closure of $\Rightarrow$, i.e. given a set of strings $\alpha_1, \ldots, \alpha_n$ such that $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ we write $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$. The sequence of strings $\alpha_1, \ldots, \alpha_n$ together with the rules applied for transforming one string into the other is called a *derivation*.

The generalization of the derivation concept for the case of synchronous context free grammars (SCFG) is straightforward, considering pairs of strings in the source and target alphabets instead of single strings. When dealing with weighted grammars, we can compute a cost for each derivation according to the scoring model. In this way we can sort the derivations for a string(-pair) according to their costs. We can then talk about e.g. an n-best list of derivations.

Note that a derivation contains smaller sub-derivations. One can define these sub-derivations to be just a subsequence of intermediate strings(-pairs) with the associated rules for the transformations. For our purposes we will need a slightly more general concept. Assume we have a derivation of the form $\alpha_1 X \alpha_2 \overset{*}{\Rightarrow} \gamma_1 \beta \gamma_2$ with X a non-terminal symbol, α_1, α_2, β, γ_1 and γ_2 arbitrary strings of terminal and non-terminal symbols, such that $\alpha_1 \overset{*}{\Rightarrow} \gamma_1$, $\alpha_2 \overset{*}{\Rightarrow} \gamma_2$ and $X \overset{*}{\Rightarrow} \beta$. We will consider $X \overset{*}{\Rightarrow} \beta$ to be a sub-derivation of the original derivation. Note that under a strictly formal point of view, the order of application of the rules may be different as in the original derivation, but the end result will be the same.

3.2 Hypergraphs

A *hypergraph* is a generalization of the concept of graph where the edges (now called hyperedges) may connect several nodes (hypernodes) at the same time. Although the definition is quite general, we will only make use of hypergraphs where a hyperedge connects a non-empty set of hypernodes to a single hypernode.

We will use hypergraphs as a representation of derivations for a given (S)CFG. Given a derivation d, each non-terminal appearing in any string in d will be represented by a hypernode. Each ingoing arc represents the rule with which the corresponding non-terminal was substituted. An example representation of an hypergraph is shown in Figure 3.2.

By sharing hypernodes and hyperedges among different derivations, a hypergraph provides a compact representation of the parsing space of a grammar.

When considering the grammar defined for the hierarchical model, each hyperedge will be associated with a rule in the source part of the grammar, together with *all* the possible

translations of the hierarchical phrase. The hypernodes will correspond to parses of substrings of the input sentence starting with a non-terminal symbol (if a substring has different parses starting with different non-terminals there will be several hypernodes). When (recursively) following an incoming hyperedge, a hypernode also *implicitly* represents a set of parses of the source sentence and the associated translations. This correspondence will only be made explicit when adding the language model information and, of course, when generating the final translation.

We will use the following terminology: If a hyperedge has a hypernode as goal node, the hypernode has an incoming hyperedge. The other hypernodes will then be the predecessors along the hyperedge. Going back to the example in Figure 3.2, the node labelled with S has an incoming hyperedge, let us call it e. The node S has two predecessors along hyperedge e, namely the nodes A and B. We denote the number of predecessors along a hyperedge as the arity of the hyperedge and will be written in equations with a pair of vertical bars $|\cdot|$. In the example, $|e| = 2$.

3.3 Compact Representation of Derivations

Using the representation of the parsing space as a hypergraph and the recursive structure of derivations (and subderivations), we can introduce a compact representation for derivations, inspired by Huang and Chiang (2005). A derivation, associated with a hypernode, will be represented by a triplet $d = (e, r, \mathbf{j})$, composed by a hyperedge identifier e, a numerical index r, corresponding to the target part for the rules associated with this hyperedge and an $|e|$-dimensional vector $\mathbf{j}$ which references the n-best derivations in the predecessor hypernodes of hyperedge e.

Each hypernode has a non-terminal symbol associated with it. The symbol associated with the head of the hyperedge e will be a string composed of a single symbol and will be the start of the derivation represented by d. The hyperedges represent a whole set of rules which share the same source part. The combination of hyperedge e and index r allows us to uniquely identify the rule used in the first step of the derivation. We will denote this rule as the *top rule* of the derivation. The resulting string will have a total of $|e|$ non-terminal symbols. For each of these non-terminal symbols we can extract the sub-derivations indexed by the vector $\mathbf{j}$ from the corresponding predecessor node of e. We will consider these to be sub-derivations of d, in the way explained above. In this way, such a triplet defines a complete derivation starting from a non-terminal symbol and ending in a pair of strings containing only terminal symbols.

3.4 A Note on Notation

Given a derivation d, represented as a triplet, we will use the notation $d[e]$ to address the corresponding hyperedge, $d[r]$ for accessing the index of the rule and $d[\mathbf{j}]$ for the predecessors. $\sigma(d)$ will be the source string associated with the derivation and similarly $\tau(d)$ the target string. $t(d)$ will be the top symbol(s) in the derivation (corresponding to the left-hand side of $d[r]$).

$\pi(e,i)$ is the i-th predecessor hypernode along hyperedge e. The set of incoming hyperedges of hypernode h will be $E(h)$. The head of a hyperedge will be denoted with $\overrightarrow{e}$.

$\mathbf{1}_D$ will be a D-dimensional vector with all its elements equal to 1 and $\mathbf{u}_i$ will be a unit vector, i.e. a vector with all elements equal to 0 except the one at position i, which has a

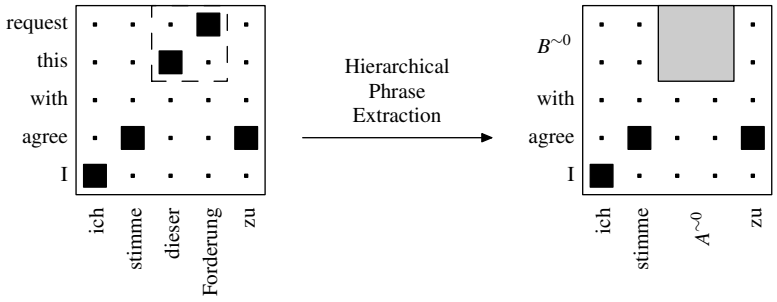


Fig. 2 Illustration of the hierarchical phrase extraction procedure. The subphrase ⟨dieser Forderung, this request⟩ is suppressed from the larger phrase and produces a gap.

value of 1. In order to reduce the clutter, we will leave the dimensionality of the vector out of the symbol, as it will be clear in each case which one is correct.

4 Hierarchical Phrase-based Translation

The intuitive concept of the hierarchical phrase-based approach is to generalize the translation units used in the phrase-based approach by allowing gaps in the phrases. This approach tries to capture long range dependencies, which may be important for generating a correct translation. The following sentence pair of a German to English translation task is an example of such a situation:

German: Die Kommission schlägt vor, die Fristen zu verkürzen, und ich *stimme* dieser Forderung *zu*.
English: The Commission suggests shorter deadlines, and I *agree* with this request.

In German, the verb "zustimmen" (meaning "agree") is separated in two parts in certain contexts. In the above example the verb is split in "stimmen" and "zu" and the construction "dieser Forderung" (meaning "this request") appears between them. However, both parts are important to produce the correct translation "agree", as there are other verbs with the same main part "stimmen" but with different additional particles, which then have different meanings. The standard phrase-based system, when seeing this training example, would memorize the complete phrase, without generalizing. In contrast, the hierarchical phrase-based approach is able to learn that the construction "ich stimme ... zu" is translated with "I agree with ...". It also learns that the translation of the gap in the German phrase is translated in the position of the gap in the English phrase. Additionally, the hierarchical approach will allow the incorporation of reordering information into a consistent statistical framework.

4.1 Hierarchical Phrases

As in the case of standard phrase-based translation, starting point for extraction is a word-aligned bilingual corpus. We will then look for phrases that contain smaller subphrases and produce gaps for them. The process is illustrated in Figure 2.

The model is formalized as a synchronous contex-free grammar. We will denote with $\mathcal{F}$ the set of words (*alphabet*, using the formal language theoretical terminology) of the source language and likewise let $\mathcal{E}$ be the alphabet of the target language. The sets of non-terminals will be denoted as $\mathcal{N}_f$ and $\mathcal{N}_e$, disjunct with $\mathcal{F}$ and $\mathcal{E}$, respectively. In the most general formulation a hierarchical rule has the form

$$(A,B) \rightarrow \langle \delta, \gamma, \sim \rangle \tag{1}$$

with $A \in \mathcal{N}_f$, $B \in \mathcal{N}_e$, $\delta \in (\mathcal{F} \cup \mathcal{N}_f)^+$, $\gamma \in (\mathcal{E} \cup \mathcal{N}_e)^+$ and $\sim$ a one-to-one relation between the non-terminals in α and β. This relation specifies how the translation process should proceed, i.e. if the non-terminal X in the source side is related to non-terminal Y in the target side, the translation of the text substituted for X is substituted for non-terminal Y.

In the baseline approach to hierarchical phrase-based translation, the identity of the non-terminals does not have any significance beyond serving as placeholders for the gaps in the phrases. We will adopt the convention of using A for the non-terminals in the source side and B for the target side.

Usual practice is to not differentiate between non-terminals in the source and target side. If two non-terminals are bound via the $\sim$ relation, they must be the same by definition. We will however depart from this convention and stay with the more general definition, allowing different non-terminals to be bound via $\sim$. Note that this does not increase the generation power of the grammar[2], but can ease the formalization when considering extensions where the identity of the non-terminals plays a role on the translation process.

The number of non-terminals in the right hand side of the rule is usually restricted to a maximum of two. In this way we have three types of rules. Without non-terminal symbols

$$(A,B) \rightarrow \langle \alpha, \beta \rangle, \tag{2}$$

with one non-terminal

$$(A,B) \rightarrow \langle \alpha_1 A \, \alpha_2, \beta_1 B \beta_2 \rangle, \tag{3}$$

and with two non-terminals

$$(A,B) \rightarrow \langle \alpha_1 A \, \alpha_2 A \, \alpha_3, \beta_1 B \beta_2 B \beta_3, \sim \rangle, \tag{4}$$

where $\alpha, \alpha_i \in \mathcal{F}^\star$ and $\beta, \beta_i \in \mathcal{E}^\star$. Note that for the rules without or with only one non-terminal we omitted the $\sim$ relationship as it is clear for these type of rules.

Normal convention is to write the rules in a more compact notation by specifying the $\sim$ relation between the non-terminals directly in the right-hand side of the rule, as a superindex of the non-terminals, e.g.

$$(A,B) \rightarrow \langle \text{weil andere } A^{\sim 0} \text{ nicht } A^{\sim 1} \text{ haben}, \text{because others have not } B^{\sim 1} B^{\sim 0} \rangle. \tag{5}$$

This rule shows an example where the verb in German (which would appear in the gap $A^{\sim 1}$) is reordered when translating into English. The rule in the example discussed above will be written as

$$(A,B) \rightarrow \langle \text{ich stimme } A^{\sim 0} \text{ zu}, \text{I agree with } B^{\sim 0} \rangle. \tag{6}$$

Non-terminals having the same superindex are bound via the $\sim$ relation.

Formally, the set of hierarchical phrases extracted from a word aligned sentence pair is best expressed in a recursive manner. Given a source sentence f_1^J, a target sentence e_1^I, an

[2] Consider pairs of symbols as new non-terminals to obtain a grammar with the terminal-bound restriction

alignment $\mathcal{A}$ between them and N the maximum number of gaps allowed, we can define the set of hierarchical phrases $\mathcal{H}(f_1^J, e_1^I, \mathcal{A})$ as

$$\mathcal{H}(f_1^J, e_1^I, \mathcal{A}) = \bigcup_{n=0}^{N} \mathcal{H}_n(f_1^J, e_1^I, \mathcal{A}), \tag{7}$$

where the $\mathcal{H}_n$ are the subsets of hierarchical phrases with n gaps. For $n = 0$ the set $\mathcal{H}_0$ corresponds to the set of standard phrases

$$\begin{aligned}\mathcal{H}_0(f_1^J, e_1^I, \mathcal{A}) = \big\{ (A,B) \to \langle f_{j_1}^{j_2}, e_{i_1}^{i_2} \rangle \mid j_1, j_2, i_1, i_2 \text{ s.t.} \\ \forall (j,i) \in \mathcal{A} : j_1 \le j \le j_2 \Leftrightarrow i_1 \le i \le i_2 \\ \wedge \exists (j,i) \in \mathcal{A} : (j_1 \le j \le j_2 \wedge i_1 \le i \le i_2) \big\}.\end{aligned} \tag{8}$$

We will denote the rules in this set $\mathcal{H}_0$ as *lexical rules*, i.e. rules where the right-hand side does not include any non-terminal symbol. We then proceed to define the following sets in a recursive manner

$$\begin{aligned}\mathcal{H}_n(f_1^J, e_1^I, \mathcal{A}) = \Big\{ (A,B) \to \langle \alpha A^{\sim n} \beta, \delta B^{\sim n} \gamma \rangle \mid \alpha, \beta \in (\mathcal{F} \cup \mathcal{N}_f)^\star, \delta, \gamma \in (\mathcal{E} \cup \mathcal{N}_e)^\star \\ \wedge \exists j_1, j_2, i_1, i_2 : j_1 < j_2, i_1 < i_2 : \\ \Big((A,B) \to \langle \alpha f_{j_1}^{j_2} \beta, \delta e_{i_1}^{i_2} \gamma \rangle \in \mathcal{H}_{n-1}(f_1^J, e_1^I, \mathcal{A}) \\ \wedge (A,B) \to \langle f_{j_1}^{j_2}, e_{i_1}^{i_2} \rangle \in \mathcal{H}_0(f_1^J, e_1^I, \mathcal{A}) \Big) \Big\}\end{aligned} \tag{9}$$

The total set of hierarchical phrases extracted from a parallel corpus is the union of the hierarchical phrases extracted from each of its sentences. We will denote this set simply as $\mathcal{H}$.

It is common practice to include two additional rules to the set $\mathcal{H}$

$$(S_A, S_B) \to \langle S_A^{\sim 0} A^{\sim 1}, S_B^{\sim 0} B^{\sim 1} \rangle \tag{10}$$

$$(S_A, S_B) \to \langle A^{\sim 0}, B^{\sim 0} \rangle \tag{11}$$

where S_A and S_B are the initial symbols in the grammar for the source and target sides, respectively. Rule (10), usually denoted "glue rule", allows the concatenation of hierarchical phrases in a manner similar to monotonic phrase-based translation. Rule (11) allows the substitution of the initial symbols of the grammar with the generic non-terminals.

The fully defined grammar is the tuple

$$\big(\mathcal{F}, \mathcal{E}, \{S_A, A\}, \{S_B, B\}, \mathcal{H}, (S_A, S_B)\big). \tag{12}$$

These equations are the ones guiding the extraction process as implemented e.g. in the open source hierarchical toolkit Jane (Vilar et al, 2010). For practical purposes, additional restrictions are usually added to the rule set, like for example a maximum length of the extracted rules. In addition, in order to increase translation efficiency by eliminating ambiguity, no adjacent non-terminals are allowed in the source side of the grammar.

4.2 Log-linear Modelling for Machine Translation

We have introduced the structural characteristics of the hierarchical phrase-based model, but we have not defined any way to assign probabilities to the translations produced by the model. Current state-of-the art statistical machine translation systems use a log-linear model to compute such probabilities. Given a source sentence f_1^J, the probability that it is translated as sentence e_1^I is given by the expression

$$p(e_1^I|f_1^J) = \frac{\exp\left(\sum_{m=1}^M \lambda_m h_m(f_1^J, e_1^I)\right)}{\sum_{\tilde{e}_1^I} \exp\left(\sum_{m=1}^M \lambda_m h_m(f_1^J, \tilde{e}_1^I)\right)}. \tag{13}$$

The $h_m(f_1^J, e_1^I)$ in Equation (13) constitute a set of *M feature functions*, each of which has an associated *scaling factor* λ_m.

Bayes decision rule states that we should select the translation $\hat{e}_1^I(f_1^J)$ which maximizes this probability:

$$f_1^J \to \hat{e}_1^{\hat{I}}(f_1^J) = \operatorname*{argmax}_{e_1^I} \left\{ p(e_1^I|f_1^J) \right\} \tag{14}$$

$$= \operatorname*{argmin}_{e_1^I} \left\{ \sum_{m=1}^M \left(-\lambda_m h_m(f_1^J, e_1^I)\right) \right\}. \tag{15}$$

The transformation from Equation (14) to Equation (15) involves just basic arithmetic transformations and the fact that the denominator in Equation (13) is a normalization factor which is independent of the translation e_1^I. In this way it can be suppressed when searching for the best translation. We will speak of the minimizing quantity in Equation (15) as the *cost* of a translation.

In practice, an approximation to Equation 15 is used. We expand the definition of the cost function to include the derivations in the grammar by adding an additional term to each of the feature functions. The selected translation is the one associated with the derivation with minimum cost. The resulting decision rule is

$$f_1^J \to \hat{e}_1^{\hat{I}}(f_1^J) = \operatorname*{argmin}_{e_1^I} \left\{ \min_{\substack{d:\sigma(d)=f_1^J \\ \tau(d)=e_1^I}} \left\{ \sum_{m=1}^M \left(-\lambda_m h_m(f_1^J, e_1^I, d)\right) \right\} \right\}. \tag{16}$$

This approximation is done mainly for practical reasons. In order to carry out the exact minimization as given in Equation 15 we should sum over all possible derivations, which is often computationally very costly (Arun et al, 2009; Li et al, 2009b).

In our system we include a standard set of feature functions:

- Translation (log-)probabilities in source-to-target and target-to-source directions.
- IBM-1 like word-level translation probabilities computed at the phrase level, also in both translation directions.
- Different word and phrase penalties.
- An n-gram language model.

The scaling factors λ_m are selected by optimizing some criterion on the training data. Current systems directly optimize the performance on a development set, i.e. a held-out subset of the training data, and thus the training criterion better reflects the evaluation criterion (Och, 2003). Usual criteria for training the scaling factors are the BLEU score (Papineni et al, 2002) and the TER metric (Snover et al, 2006). In our case, all experiments are optimized with respect to BLEU.

4.3 The Search Problem

We denote the task of finding the translation according to Equation (16) as *search problem*. For the hierarchical translation model this problem is usually formulated as a deductive system (Chiang, 2007; Lopez, 2009). In this section we will introduce a formalization of the search as a dynamic programming problem. This is more consistent with a big part of the available literature about statistical machine translation, see for example (Tillmann and Ney, 2003; Koehn, 2004; Zens and Ney, 2008).

The set of dynamic programming equations will follow a similar structure as the equations for the Cocke-Younger-Kasami (CYK) parsing algorithm. We will introduce an auxiliary quantity $Q(j_1, j_2, (A,B), \tilde{e})$, which represents the cost of the best translation covering the span $f_{j_1}^{j_2}$, having the non-terminal symbols (A,B) as start symbols for this sub-parse and the language model context $\tilde{e}$. As discussed in Section 4.2 the cost of a translation is a log-linear combination of different models, and thus

$$Q(j_1, j_2, (A,B), \tilde{e}) = \min_{\substack{d:\sigma(d)=f_{j_1}^{j_2} \\ h(\tau(d))=\tilde{e} \\ t(d)=(A,B)}} \left\{ \sum_{m=1}^{M} \left(-\lambda_m h_m(f_{j_1}^{j_2}, \tau(d), d) \right) \right\} . \tag{17}$$

The h function used in Equation (17) computes the bidirectional context of a given string. It will be defined in detail in Equation (23). Suffice to say at this point that this information is the language model state and will be necessary at a later stage for computing the language model score. The cost of the best translation will be the lowest value of the form

$$Q(1, J, (S_A, S_B), \cdot) , \tag{18}$$

and the best translation can then be obtained by backtracking through the dynamic programming table.

We can differentiate two type of features, depending if they can be computed for each rule separately or not. In our system, all the features listed in Section 4.2 except the language model can be computed at rule level. We will define the cost of a rule as the sum of these features functions, weighted with the corresponding scaling factors of the log-linear model. We will denote it as *translation costs*, in symbols $c_T(r)$ for every rule r of the hierarchical model. Denoting with d_r a derivation consisting only of rule r and assuming the language model is the M-th model in the log-linear combination, we can write

$$c_T(r) = \sum_{m=1}^{M-1} \left(-\lambda_m h_m(\sigma(d_r), \tau(d_r), d_r) \right) \tag{19}$$

The language model, because of its context dependencies which expand beyond phrase-boundaries, has to be handled separately.

In order to simplify the exposition, we will concentrate on the standard case where the rules have a maximum of two gaps. As a first step, and merely for clarity, we will split the main quantity into three separate equations

$$Q(j_1, j_2, (A,B), \tilde{e}) = \min \{ Q_0(j_1, j_2, (A,B), \tilde{e}), Q_1(j_1, j_2, (A,B), \tilde{e}), Q_2(j_1, j_2, (A,B), \tilde{e}) \} . \tag{20}$$

Each of the additional Q_n functions corresponds to the same quantity as the original function Q, but with the restriction that the last rule applied has exactly n terminals. The case of $n = 0$ corresponds to the initial phrases and can be considered as an initialization step

$$Q_0(j_1, j_2, (A,B), \tilde{e}) = \min_{\substack{r \in \mathcal{H}: r=(A,B) \to \left\langle f_{j_1}^{j_2}, \tilde{e}' \right\rangle, \\ h(\tilde{e}')=\tilde{e}}} \left\{ c_T(r) + c_{LM}(\tilde{e}') \right\} . \tag{21}$$

In this equation, the rule r associates the sequence of source words $f_{j_1}^{j_2}$ with the translation $\tilde{e}'$. c_{LM} corresponds to the language model cost associated with this translation, taking into account the words that can be computed with an n-gram language model. More formally, if we are considering an n-gram language model

$$c_{LM}(e_1^m) = -\lambda_{LM} \cdot \sum_{\substack{n \leq i \leq m \\ \diamond \notin e_{i-n+1}^{i}}} \left[\log p(e_i | e_{i-n+1}^{i-1}) \right] , \tag{22}$$

with λ_{LM} the scaling factor associated with the language model in the log-linear combination. The need for handling a special $\diamond$ symbol as a separate case will become apparent when we discuss the functions Q_1 and Q_2, below.

With $\tilde{e}$ we represent the language model context that we need for further computations. It corresponds with the left-most and right-most words of the generated translation. We must keep track of the left-most part of the translation, as these words still need to be scored by the language model when expanding the combining this derivation with other ones as the translation process continues. The right-most part is needed to supply the necessary context information for the correct scoring of newly produced words by the language model. Note that the two groups of words are not necessarily disjunct. Both parts are stored in a single string, separated with the special *omit symbol* $\diamond$. Thus $\tilde{e} \in (\mathcal{E} \cup \{\diamond\})^\star$, however this special omit symbol may appear only once in the string.

The h function encodes this context information[3] given a string in the target language. The function is defined as (Chiang, 2007)

$$h(e_1^m) = \begin{cases} e_1^m & \text{if } m < n \\ e_1^{n-1} \diamond e_{m-n+2}^m & \text{otherwise} \end{cases} , \tag{23}$$

assuming a language model of order n. If the string given to this function has a length smaller than the order of the language model, it remains unchanged. However, if the length is bigger than the order of the language model, the h function eliminates those words for which the language model score has already been computed and which are not needed for future language model computations. Example applications of the h function are shown in Figure 3.

We now proceed to discuss the case when we deal with more than one gap. The definition of the Q_1 function is as follows

$$Q_1(j_1, j_2, (A,B), \tilde{e}) = \min_{\substack{r \in \mathcal{H}: r=(A,B) \to \left\langle u_1 A^{\sim 1} u_2, v_1 B^{\sim 1} v_2 \right\rangle \\ j_1', j_2': u_1 f_{j_1'}^{j_2'} u_2 = f_{j_1}^{j_2} \\ \tilde{e}': h(v_1 \tilde{e}' v_2) = \tilde{e}}} \left\{ Q(j_1', j_2', (A,B), \tilde{e}') + c_T(r) + c_{LM}\left(v_1 \tilde{e}' v_2\right) \right\} . \tag{24}$$

[3] h for language model *history*.

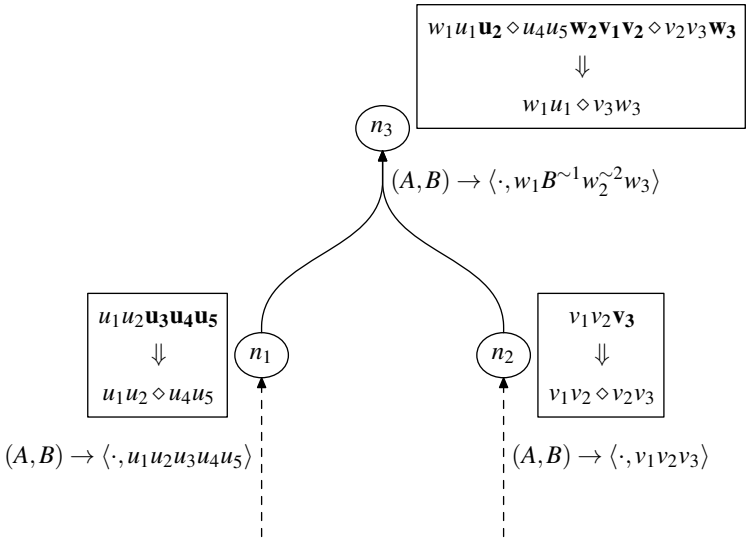


Fig. 3 Language model context computation example, assuming a 3-gram. Each hyperedge has a rule associated with it (only the target part of the rules is shown). The boxes beside each hypernode show first the string used as an argument to the c_{LM} function. The words for which the LM score can be computed are shown in bold. Afterwards the result of applying the h function to this string is given.

In this equation there are several indices over which the minimization is carried over. We have to maximize over two source indices j_1' and j_2' that represent an already translated sub-sequence of the source sentence. We also have to minimize over the rules that we can use for extending the given translation. The cost of the translation will be the sum of the cost of the rule, the language model cost and the cost of the best translation of the segment $f_{j_1}^{j_2}$, which is given in a recursive manner by the function Q. Note that we use the general Q function, not a version with subindices, so that we can model the interaction between rules with an arbitrary number of gaps. Note also that the argument for the c_{LM} function includes the context information $\tilde{e}'$ of a previously computed hypothesis. This context information may contain the special language model symbol $\diamond$, which is the reason we had to consider this case in the definition of the function c_{LM} in Equation (22) (see also Figure 3).

The formalization for a higher number of gaps is then quite straightforward, although the notation may become a bit awkward. The definition of the function Q_2 is

$$Q_2(j_1, j_2, (A,B), \tilde{e}) = \max_{\substack{r \in \mathcal{H}: r = (A,B) \rightarrow \langle u_1 A^{\sim 1} u_2 A^{\sim 2} u_3, v_1 B^{\sim \rho_1} v_2 B^{\sim \rho_2} v_3 \rangle \\ j_1', j_2', j_1'', j_2'': u_1 f_{j_1'}^{j_2'} u_2 f_{j_1''}^{j_2''} u_3 = f_{j_1}^{j_2} \\ \tilde{e}_1, \tilde{e}_2: h(v_1 \tilde{e}_{\rho_1} v_2 \tilde{e}_{\rho_2} v_3) = \tilde{e}}} \Big\{ Q(j_1', j_2', (A,B), \tilde{e}_1) + Q(j_1'', j_2'', (A,B), \tilde{e}_2) + c_T(r) + c_{LM}\left(v_1 \tilde{e}_{\rho_1} v_2 \tilde{e}_{\rho_2} v_3\right) \Big\}. \tag{25}$$

This equation follows the same basic structure as (24), but with a search over two gaps, represented by two pairs of indices (j_1', j_2') and (j_1'', j_2''). In the same way the function Q is recursively called two times when computing the cost of the hypothesis. For the case of two non-terminals, we have to consider that the translations for these non-terminals may be

reordered in the rule. This is expressed in Equation (25) with the ρ notation which reflects the binding of non-terminals by the $\sim$ correspondence. If $\rho_1 = 1$, then the first non-terminal in the target part of the rule is bound to the first non-terminal in the source part. If this is not the case, then $\rho_1 = 2$. The same holds for ρ_2, but of course $\rho_1 \neq \rho_2$.

Following the same schema, we can generalize these dynamic programming equations for an arbitrary number of non-terminals.

5 Adapting a Parsing Algorithm

Having defined the translation problem, we will now discuss how to efficiently carry out the generation process. The formalization of the hierarchical phrase-based translation approach is a parallel context-free grammar and, as such, the translation process can be thought of as a parsing problem. Given an input sentence in the source language, we parse this sentence using the source language part of the parallel grammar defined by the hierarchical rules. Because each source rule has an associated rule in the target language, we can construct a parse tree in the target language given a tree in the source language. The yield of this tree will be the translation of the input sentence.

In this section we will discuss the principles of an efficient monolingual parsing algorithm that will be applied as the first step of the translation process.

5.1 The Language Model Problem

If the cost of a translation would not include the score computed by the language model, we could use one of the probabilistic extensions of well known parsing algorithms and extract the translation as the yield of the parse tree induced in the target side, as pointed out above. However, the language model, due to its context dependencies, interferes with this simple approach. This model depends not only on the individual rules that are used in the translation, but also on the way they are combined. For computing the probability of a word, the language model takes into account the context this word appears in, and this may well extend beyond the limits of the rule producing this word. This fact interferes with most parsing algorithms, as they only include information about the non-terminal that spans some part of the input sentence.

One straightforward way for including the language model information would be to generate an n-best list of possible translations, compute the language model score on these translations and select the one with the best combined score. This approach, however, is not reasonable in practice. The n-best list is only a crude approximation to the whole search space. As such, many good translations may be skipped in the generation process of the n-best list because the translation models alone are not accurate enough to identify them. The rescoring process with the LM will then not be able to recover from this early search error. Of course, this effect may be alleviated by making the size of the n-best list bigger, but for obtaining good results, this size would need to be too large for efficient computation.

A more justified approach is to include the language model computation already at generation time. In order to do this, and also to efficiently keep track of the big number of translation alternatives (note that several rules may share the same source part but have a different target part), we will construct a hypergraph which will represent the set of possible parse trees for the parallel grammar, as discussed in Section 3.

5.2 Adapting the CYK+ Algorithm

The CYK+ algorithm (Chappelier and Rajman, 1998) is an extension of the well-known CYK algorithm which relaxes the requirement for the grammar to be in Chomsky normal form. This algorithm works with a grammar which consists of *non partially lexicalized rules*, i.e. grammars where terminals can only appear in rules of the form $X \rightarrow w_1 w_2 \dots w_n$, with $w_1, \dots, w_n$ terminal symbols. A simple transformation adapts the grammar of the hierarchical model to this form.

Given a (possibly transformed) grammar composed only of non partially lexicalized rules, the CYK+ algorithm proceeds in a similar way as the standard CYK algorithm. It is an algorithm based on the dynamic programming paradigm which fills a triangular chart indexed by a "starting position" and the length of a substring starting at this position. In the standard CYK algorithm, each cell in the chart stores a list of non-terminals which can parse the corresponding substring. This list is also present in the CYK+ algorithm and is called a *type-1 list*. Additionally, we store in each chart cell a second list, called *type-2 list*, which is composed of items of the form $\alpha\cdot$, with α a string of non-terminal symbols that represent partial parses of the corresponding substring, and for which there are rules in the grammar whose right hand side starts with the string α.

More formally, given the input string $w_1 \dots w_N$, the non-terminal symbol X appears in the type-1 list of the cell (k,l) if and only if $X \stackrel{*}{\Rightarrow} w_k \dots w_{k+l-1}$ and the item $\alpha\cdot$, appears in the type-2 list of the cell (k,l) if and only if $\alpha \stackrel{*}{\Rightarrow} w_k \dots w_{k+l-1}$ and there exists a rule of the form $A \rightarrow \alpha\beta$, where β is a non-empty string of non-terminal symbols. The procedure for filling the chart follows the same pattern as the standard CYK parsing.

The discussion of the CYK+ algorithm up to this point has been centered in the monolingual parsing problem and followed the description given in (Chappelier and Rajman, 1998). However, for the task of hierarchical phrase-based translation, we must take some additional aspects into account. As pointed out above, in order to better integrate the language model, we are interested in the representation of the parsing space as a hypergraph. Furthermore, we must keep track of the translations associated with the rules in our bilingual grammar. We will extend the CYK+ algorithm so that these two issues are observed.

Recall from Section 3.2 that each hyperedge will be associated to a rule in the source part of the grammar, together with the translations of the hierarchical phrase. The hypernodes will correspond to parses of substrings of the input sentence starting with a non-terminal symbol. Taking this into account, we will expand the lists that are used in the CYK+ algorithm. The elements of the type-1 lists will be pairs composed of a non-terminal and a hypernode. Note however that only one pair with a given non-terminal may appear in a type-1 list. Or put it in another way, a hypernode will be uniquely identified by the cell it resides and a non-terminal. The elements of the type-2 lists will also be pairs. The first element will be a partial rule application, as in the original CYK+ algorithm. The second element will be a list of hypernodes. This list will be used for adding the hyperedges when creating new hypernodes in the course of the parsing process. The hyperedges roughly correspond to adding backpointers in the standard parsing chart.

With this modifications and with the corresponding combination operations, the result of the CYK+ algorithm is a compact representation of the whole parsing space of the translation model, which also implicitly encodes all the translation alternatives. In the next two sections we will discuss how to efficiently extract the translation with the lowest cost from this representation.

6 The Cube Pruning Algorithm and Extensions

The cube pruning algorithm was first presented in (Chiang, 2007) and can be considered an adaptation of one of the n-best parsing algorithms presented in the previous paper (Huang and Chiang, 2005). Given the search space represented as a hypergraph, the algorithm is centered on the generation of n-best lists of derivations. The case of single-best translation can of course be considered a special case when $n = 1$. Due to the non-monotonicity introduced by the language model at one stage of the computation, even for the single-best case we will have to consider the generation of a n-best list and then select the best translation.

The cost of a derivation can be decomposed, in a similar way as in Section 4.3, into the translation of the top rule of the derivation, the language model cost and the cost of the sub-derivations that it includes. For the language model, however, we must consider only the words that have been produced in this derivation plus the ones in previous derivations, for which enough LM context has been made available (in the same spirit as in the dynamic programming equations above). We will denote those words with $\hbar(d)$. We have then for the cost of a derivation

$$c(d) = c_T(d[r]) + c_{LM}(\hbar(d)) + \sum_{i=1}^{|e(d)|} c(\pi(d[e],i))\,, \tag{26}$$

which corresponds to the minimizing expression in Equation (25), but formulated on the hypergraph level. $\pi(d[e],i)$ represents the predecessors along hyperedge e, as introduced in Section 3.4.

The combination of these costs is just a sum and thus monotonic. Let us ignore the contribution of the language model for the initial exposition, i.e. let us drop the term $c_{LM}(\hbar(d))$ in Equation (26). For each hyperedge, because now all the costs are only "local", the best derivation will be composed of the translation with the lowest cost and the best derivation of each of the predecessor nodes. The second-best derivation will then be *adjacent* to this one. Adjacent in this context means that the second-best derivation will differ from the first-best either by associating the second-best translation of the hyperedge or the second-best derivation of one of the predecessor hypernodes. But it will differ in *only one* of these. If it would differ in more than one at a time, the monotonicity property shows that the derivation differing in only one will have a better score.

The principle is illustrated in Figure 4. This diagram shows the process at the stage of a 3-best generation along a hyperedge with two predecessors. The first derivation to be generated is the one represented in the upper left corner, composed by combining the best derivations of each of the predecessors using the translation with the lowest cost. For taking into account all the incoming hyperedges of a hypernode, we just pool all the resulting cubes and select the best derivation at each stage.

We can formalize the cube pruning algorithm by defining a set $A_n(h)$ of *active* derivations for hypernode h at step n of the algorithm (the greyed cubes in Figure 4). The nth-best derivation $d_n(h)$ will be the one with the minimum cost in this set:

$$d_n(h) = \operatorname*{argmin}_{d \in A_n(h)} \{c(d)\}\,. \tag{27}$$

The initial set of active derivations is defined as the set of the first best derivations for each incoming hyperedge (the upper left corner in the cube), i.e.

$$A_1(h) = \bigcup_{e \in E(h)} \left\{(e, 1, \mathbf{1}_{|e|})\right\}. \tag{28}$$

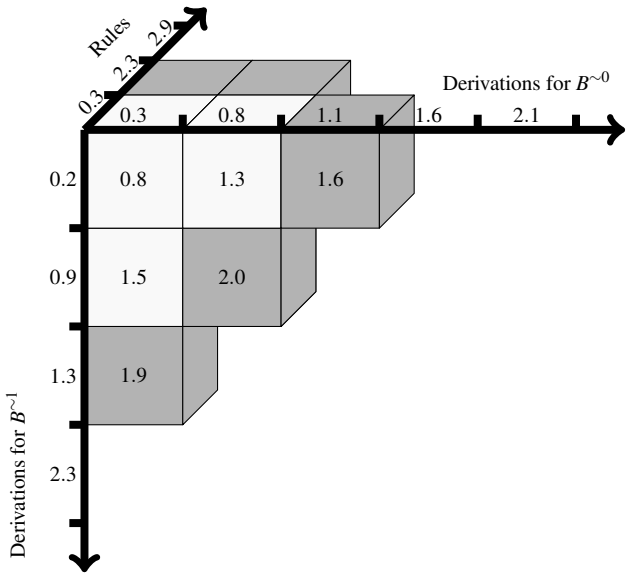


Fig. 4 Illustration of the cube pruning algorithm for a hyperedge with two predecessors. Each axis corresponds to each element that contributes to the total score: the derivations associated with each of the non-terminals and the possible rules (translations) in the hyperedge. The 3 lightly shaded cubes correspond to the 3-best derivations along the hyperedge and have been generated in order of increasing costs. The dark shaded cubes correspond to the active candidates for the next-best derivation.

The following sets are defined in a recursive manner by including the adjacent derivations of $d_n(h)$, but removing the derivations we have already generated. In order to simplify the notation we introduce an auxiliary function $g(A,d,D)$ (for *growing*) which returns the updated set of active derivations given the current one, A, the selected derivation d and the set of already generated derivations D

$$g(A,d,D) = \left(A \cup \left\{ (d[e], d[r]+1, d[\mathbf{j}]) \right\} \cup \bigcup_{i=1}^{|d[e]|} \left\{ (d[e], d[r], d[\mathbf{j}]+\mathbf{u}_i) \right\} \right) \setminus D \,. \tag{29}$$

We can then give the general definition

$$A_{n+1}(h) = g\big(A_n(h), d_n(h), \{d_i(h)\}_{i=1}^{n}\big) \,. \tag{30}$$

Here we will not discuss out-of-bound conditions (e.g. for a hypernode we may not be able to generate the desired size for the n-best list), although in a practical implementation this has of course to be taken into account.

Including the language model information breaks the monotonicity property underlying the cube pruning algorithm, as this cost depends on the identity of the different elements present in a derivation. Thus we cannot guarantee that the best derivation will be the one composed by the best rule and the best derivations of the predecessors. The language model score may alter the costs in such a way that other derivations may now have a lower total cost. A possible solution for this issue, and the one normally used in practice is actually relatively simple, and perhaps a bit "brute-force". We just ignore the non-monotonicity introduced

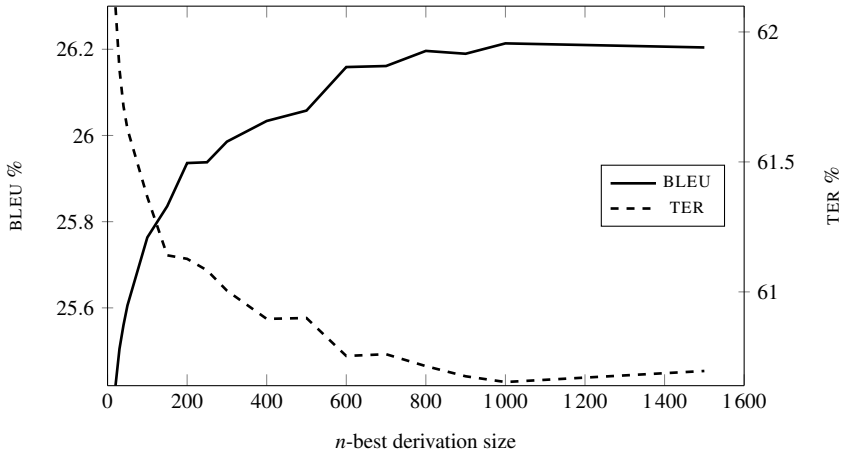


Fig. 5 Effect of n-best generation size on translation performance.

by the language model, i.e. we still apply Equations (27) to (30), but with the full-fledged cost-function. It may, and probably will be the case that the enumeration of the elements does not follow a best-score order, therefore we first store the derivations in an intermediate buffer, which will be sorted at the end of the generation procedure. There is no guarantee that we are indeed generating a true n-best list, as the non-monotonicity may mislead the algorithm and generate a sub-optimal set of derivations while ignoring the actual best set. In order to counteract this effect, we just generate a bigger list (the actual size is highly dependent on the task at hand) and from this list we select the best n entries. This is the reason why this algorithm is used also for single-best translation in spite of actually being an n-best translation algorithm.

An additional point that has to be taken into account is in which order the hypernodes are visited when generating the derivations. Because the derivations for a hypernode depend on the derivations of predecessor hypernodes, the hypergraph has to be traversed in topological order. For the standard model, this can be easily accomplished by traversing the CYK+ chart in a bottom-up manner, generating first the derivations for the generic non-terminals and afterwards for the initial non-terminals (if present).

The effect of the size of the n-best list can be seen on Figure 5, on the 2006 development corpora of the German-English WMT task (see also Section 8). As can be seen from this figure, the bigger the n-best list generated at each node, the better the BLEU and TER scores. At an n-best size of 1 000, however, a saturation point is reached and further increasing the size of the n-best lists does not show any improvement in performance.

6.1 Source Cardinality Synchronous Cube Pruning

The cube pruning algorithm presented in the preceding section has a major drawback: in each hypernode we compute a fixed number of derivations, namely the size of the goal n-best list we want to compute. One has to consider that the computation of the cost of a derivation involves, among other things, the computation of a language model score, and this is a costly operation, specially when dealing with big order n-grams trained on large

amounts of text data. In this and the next sections we will discuss two ways to reduce the absolute number of derivations we compute in the translation process, thus increasing the computational efficiency.

We will adapt the concept of *source cardinality synchronous search* (SCSS) widely used in standard phrase-based translation. The cube pruning algorithm works on the hypernode level, i.e. an n-best list is computed at each node of the hypergraph independently of the others. This means that for each set of (contiguous) source words we generate a list of derivations, given the interpretation of a hypernode within a hypergraph derived from parsing. In contrast, in SCSS for phrase-based translation, hypotheses covering the same *amount* of source words are considered as "competing" in the pruning process. In the case of the cube pruning algorithm, the pruning itself is the non-generation of derivations because of their placement in the cube of candidates.

In this case, instead of traversing the hypergraph and computing the n-best list for each hypernode, we will compute this n-best list for each cardinality C, from 1 up to the length of the source sentence. We will again have a set $A'_n(C)$ of active derivations, and the nth-best derivation will then be the one with minimum cost among them

$$d'_n(C) = \underset{d \in A'_n(C)}{\operatorname{argmin}} \{c(d) + r(d)\}\,. \tag{31}$$

This equation corresponds to Equation (27), but is formulated at the cardinality level. Also, there is an additional term in the minimization expression, namely $r(d)$. The costs of derivations covering different parts of the source sentence are now compared. There may be certain groups of words in a sentence which might have comparatively low cost to translate with respect to other groups. E.g. if the translation is not ambiguous, the costs are expected to be low. In this way the generation process may concentrate first on these easy-to-translate parts and may neglect the generation of needed derivations for the difficult parts in an early stage of the process, which may lead to search errors. This effect is known from the phrase-based approach, or more generally for those approaches where hypothesis with different coverages have to be compared. A possibility to alleviate this problem consists in computing a heuristic of the translation costs of the yet uncovered parts in the source sentence, very much in the spirit of A* search. This is known as future cost (Koehn, 2003) or rest cost (Och, 2002)[4] estimation. This last terminology will be used in the following.

For computing the rest costs, we can again differentiate between the language model cost and the translation costs. Due to the structure of the CYK+ algorithm, as in the standard CYK, the partial derivations always cover contiguous parts of the source sentence. Because the pooled cube prune algorithm is applied *after* the parsing has been completed, we can use this fact to compute the translation rest costs in an efficient way. First we note that for a given partial derivation there are at most two contiguous segments of the source sentence that still need to be translated. If we extend the CYK+ algorithm in a straightforward way to keep track of the derivation with the lower cost for a certain range of words, we can readily use this information as a lower bound of the rest costs for these source segments. The overhead for computing this heuristic is then minimal. As an additional note, this computation does not include the cost of the rule that is used for combining all the partial translations.

For computing the language model rest cost heuristic we follow the approach in (Och and Ney, 2004), where for every target word in the vocabulary the best possible cost in the language model is determined (searching over all contexts). This step can be performed

[4] The concept was introduced in (Och, 2002) but the terminology is to be found in (Zens, 2008).

beforehand and the results stored as an additional information source for the translation process. During search, for the translation alternatives we can use these optimistic estimations of the language model costs for each word to compute a heuristic for the language model rest cost.

We still have to give the expressions for the sets of active candidates in order to fully define the source cardinality synchronous cube pruning algorithm. We just need to substitute Equation (28), where the initial set of active derivations is given. The definition is quite similar, but now the union goes over all the hypernodes with the given cardinality. We will denote the cardinality associated with a hypernode h with $\mathcal{C}(h)$. We thus have

$$A'_1(C) = \bigcup_{h:\mathcal{C}(h)=C} \bigcup_{e\in E(h)} \left\{(e, 1, \mathbf{1}_{|e|})\right\}. \tag{32}$$

The general definition is then analogous to Equation (30), but on the cardinality level:

$$A'_{n+1}(C) = g\left(A'_n(C), d'_n(C), \{d'_i(C)\}_{i=1}^n\right). \tag{33}$$

6.2 Coverage Pruning

We might still be interested in controlling the amount of derivations that are generated for each hypernode, in a similar way as in the unaltered cube pruning algorithm. We will impose an upper limit N_H to the amount of derivations that can be generated for each hypernode. In this way we have a better control of the search and we can avoid the one-hypernode-takes-all effect we discussed above for the rest costs.

We only need to update the sets of active derivations. The initial one is exactly the same as in Equation (32):

$$A''_1(C) = \bigcup_{h:\mathcal{C}(()h)=C} \bigcup_{e\in E(h)} \left\{(e, 1, \mathbf{1}_{|e|})\right\}. \tag{34}$$

The general case is similar to Equation (33), but after generating a new derivation we have to check if we arrived at the upper limit N_H. If this is the case, we eliminate the derivations corresponding to this hypernode from the set of active derivations. The resulting equation is

$$A''_{n+1}(C) = \begin{cases} g\left(A''_n(C), d''_n(C), \{d''_i(C)\}_{i=1}^n\right) & \text{, if } \sum_{i=1}^n \delta\left(\overrightarrow{d_i[e]}, \overrightarrow{d_n(C)[e]}\right) < N_H \\ A''_n(C) \setminus \bigcup_{d\in A''_n(C):\overrightarrow{d[e]}=\overrightarrow{d''_n(C)[e]}} \{d\} & \text{, otherwise} \end{cases} \tag{35}$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function. It is used in this equation as a way of counting how many derivations share the same head hypernode as the derivation what was just generated.

Note that in the discussion we have always dealt with a fixed maximum amount of derivations. The algorithms can also be formulated to take a threshold into account. That is, we keep track of the derivation with the best cost. New derivations whose cost is greater than this best cost plus some given margin will then be rejected. Preliminary experiments applying this thresholding did not show any gain, therefore we chose to leave it out of the discussion.

The pseudo-code for the source cardinality synchronous cube pruning algorithm is shown in Figure 6. The algorithm receives a set of hypernodes as an argument, which in this case

will be the hypernodes corresponding to a given cardinality. We will maintain the set of generated derivations indexed by the hypernodes, in the sets D_n. The set $A''_k(C)$ will be represented at iteration k of the algorithm by the set c of possible candidates (organized as a heap, for efficiency). We additionally keep a set g of already generated derivations, in order to avoid duplicates.

Lines 3 to 9 of the algorithm serve as initialization. Lines 10 to 16 are the main part of the algorithm, where the candidate derivation of the lowest cost is selected at each iteration. This part makes use of the function PUSHSUCC[5], which mainly corresponds to the g function defined in Equation 29. The coverage pruning is performed in line 13. As a last step the generated derivations are sorted for each hypernode.

The behaviour of this algorithm is shown in Figure 7 for some representative values. As can be seen, for lower values of cardinality pruning, it is important to choose an appropriate value for coverage pruning. If too many derivations are allowed per coverage, we can see the effect of the derivation concentration in some hypernodes and the quality of the translation suffers from it. When applying a wider cardinality pruning beam the effects of coverage pruning are not so critical, but they help in achieving the same level of performance with less computational effort.

The comparison of all variants of the cube pruning algorithm in terms of translation performance depending on computation effort can be seen in Figure 8. In this graph the x-axis corresponds to the average number of derivations, or partial hypothesis using a more standard terminology, that have been computed per source word. Because the rest cost estimation can be done in a very efficient way, this is an accurate measure for comparing the performance of the algorithms. We can see that the source cardinality synchronous cube pruning itself improves the translation quality a little bit, but with nearly no improvement in computational effort. The rest cost estimation has some little effect on translation quality for a restricted search space, but it becomes minimal when a more exhaustive search is conducted. If we include coverage pruning, the translation quality is again slightly improved, but in this case with significantly less search effort.

Note that these extensions to the cube pruning algorithm may also help for the case where the number of non-terminals is increased, like for example in the SAMT model Zollmann and Venugopal (2006). In this case the size of the hypergraph is dramatically increased, as hypernodes are "replicated" in order to consider parses headed with the new non-terminals. The cost of the standard cube pruning algorithm, with a fixed size n-best list at each node, is also increased in the same way. By pooling hypernodes with the same cardinality we can have a finer control on the computational cost.

7 The Cube Growing Algorithm

Another possibility to reduce the number of derivations that are generated is to compute them on-demand, deferring the generation until the moment they are needed. The cube growing algorithm (Huang and Chiang, 2007) is a reformulation of the cube pruning algorithm following this strategy. Instead of traversing the hypergraph in a bottom-up manner, generating a fixed amount of derivations at each hypernode, the cube growing algorithm starts at the goal node. It recursively calls itself on the predecessor nodes, computing the necessary derivations on demand.

[5] The naming of this function is due to Huang and Chiang (2007). We reformulated the pseudo-code adapting it to our notation.

Input: A set N of hypernodes and the size k of the k-best list
Output: $\{D_n\}$, a set of lists with the n-best derivations for each hypernode
let $c = \text{heap}()$
let $g = \{\}$
for each $n \in N$ **do**
    let $D_n = []$
    for each incoming hyperedge e of n **do**
        add $(e, 1, \mathbf{1}_{|e|})$ to c
        add $(e, 1, \mathbf{1}_{|e|})$ to g
while $|c| > 0$ and $\sum_{n \in N} |D_n| < k$ **do**
    $d = \text{pop}(c)$
    $D_{\overrightarrow{d[e]}} = D_{\overrightarrow{d[e]}} {}^{++}[d]$
    if $\left|D_{\overrightarrow{d[e]}}\right| < N_H$ **then**
        PUSHSUCC(d, c, g)
    else
        $c = c \setminus \bigcup_{d \in c : \overrightarrow{d'[e]} = \overrightarrow{d}} \{d'\}$
for each $n \in N$ **do**
    sort D_n

// Auxiliary function
function PUSHSUCC(d, A, g)
    notation: $d = (e, r, \mathbf{j})$
    notation: $(n_1, \ldots, n_{|e|})$ = predecessor hypernodes of e
    notation: R = set of target parts associated with e
    $\mathbf{j}' = \mathbf{j}$
    for $i = 1$ **to** $|e|$ **do**
        $\mathbf{j}'[i] \mathrel{+}= 1$
        if $|D_{n_i}| \geq \mathbf{j}'[i]$ **then**
            if $(e, r, \mathbf{j}') \notin g$ **then**
                push $(e, r, \mathbf{j}')$ into A
                $g = g \cup \{(e, r, \mathbf{j}')\}$
        $\mathbf{j}'[i] \mathrel{-}= 1$
    if $r < |R|$ **then**
        if $(e, r+1, \mathbf{j}) \notin g$ **then**
            push $(e, r+1, \mathbf{j})$ into A
            $g = g \cup \{(e, r+1, \mathbf{j})\}$

Fig. 6 The source cardinality synchronous (SCS) pooled cube pruning algorithm, including coverage pruning.

An illustration of the principle is presented in Figure 9. Again to simplify the exposition, let us ignore the language model score so that we can rely on the same monotonicity property we presented for the cube pruning algorithm. The figure shows a possible state in a 5-best generation at node n_1. For constructing this derivation we need the black coloured derivations shown in the rest of the nodes n_2 to n_9. Note that none of them is the 5th-best derivation in its corresponding hypernode. The gray coloured derivations needed to be computed in order to assure that order of the derivations is correct. Following this reasoning we can arrive at an important observation: the amount of possibilities to combine different derivations of hypernodes further down the hypergraph allow to compute a relatively large

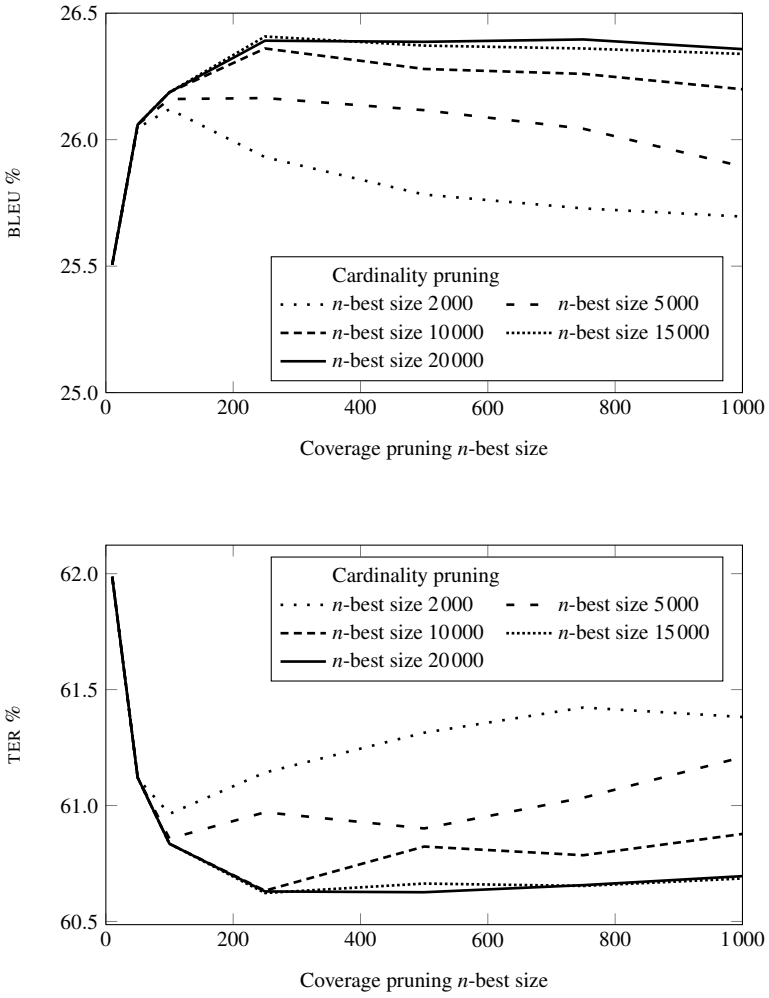


Fig. 7 Results for the source cardinality synchronous cube pruning algorithm including coverage pruning. The cardinality pruning n-best size limits the amount of derivations that are computed for any cardinality. The coverage pruning n-best size limits the number of derivations in each hypernode.

n-best list of derivations for the upper nodes in the hypergraph, reducing the number of derivations that are needed in the lower hypernodes.

The formalization of the cube growing algorithm is then the same as for the cube pruning algorithm, i.e. Equations (27) to (30). The main difference is how the computation of the sets $A_n(h)$ of active hypotheses is organized. In cube pruning they are computed as a whole, in a top-down manner. In cube growing, except for the goal hypernode, in most cases we will not compute a whole set of active derivations. The generation of derivations of the different sets is also intertwined. Huang and Chiang (2007) give an algorithmic description of the search organization.

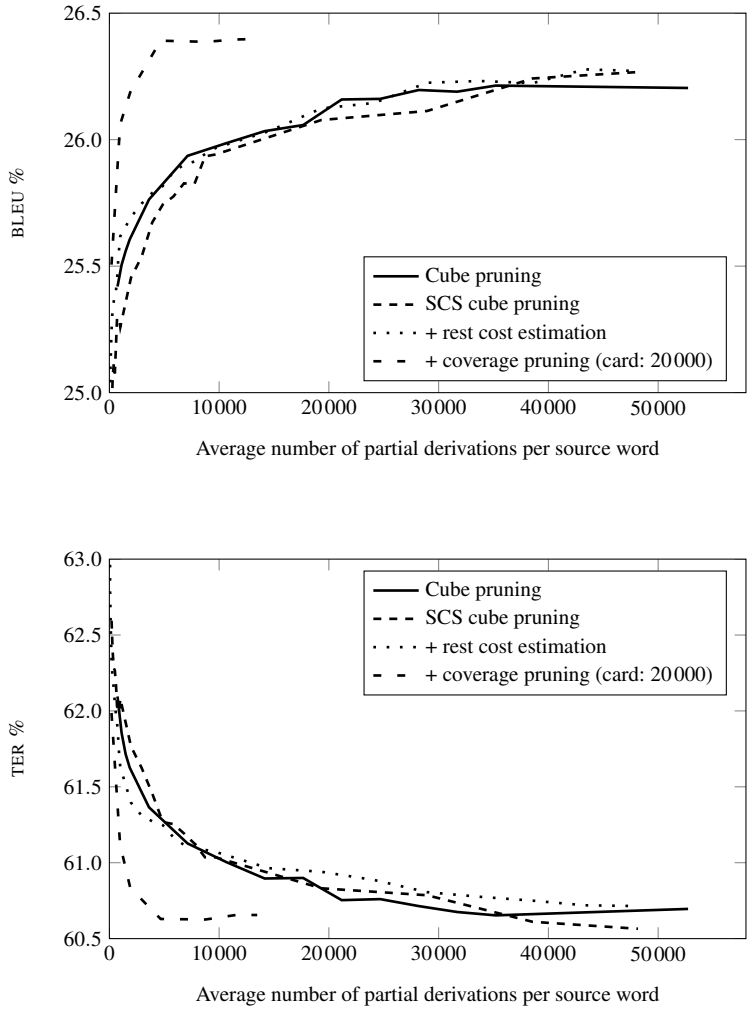


Fig. 8 Comparison of the different variations of the cube pruning algorithm. "SCS cube pruning" stands for source cardinality synchronous cube pruning. The performance is plotted against the search effort. Depending on the algorithm, the search algorithm is controlled by varying different n-best sizes: for each hypernode for cube pruning, for each cardinality for SCS cube pruning (with and without rest cost estimation), and again for each hypernode when adding coverage pruning, while maintaining the n-best size at 20 000 for each cardinality.

7.1 LM Heuristics

When including the language model, the monotonicity property is once more lost and we have to pay special attention in order to minimize the amount of search errors. The simple strategy of generating a bigger n-best list, as applied for the case of cube pruning, cannot be used in this case, since the advantage of the on-demand computation would be lost. Instead, we will introduce an additional, intermediate buffer where we store the derivations while we generate them. Once we are confident that no better derivations can be generated, we will

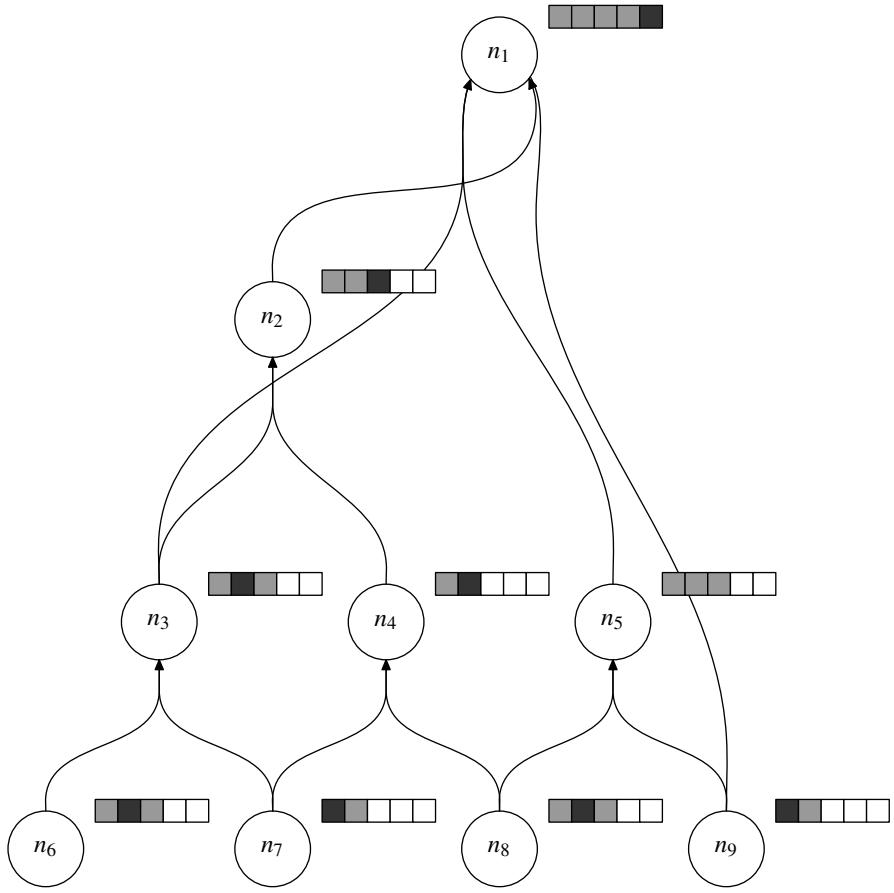


Fig. 9 Illustration of the principle behind the cube growing algorithm. Each list associated with an hypernode represents an n-best list. The dark shaded elements are the derivations that are needed for the current best derivation at the goal hypernode (n_1). The lightly shaded elements had to be computed in order to make sure that no best derivation exists. It can be observed that a full n-best list had to be computed only for the goal hypernode.

consider a derivation in this buffer to be "accepted" and will be added to the corresponding n-best list. The key question is thus how to decide that none of the derivations still to be generated will have a lower cost that the ones we have already produced.

We will compute an estimation of the LM score for the still to be computed derivations in the form of an approximate *heuristic*. One key issue for practical application is that the heuristic computation must be efficient. If too much time is spent on computing the heuristic, the gains of the lazy evaluation can be overcome by this computation time. In the extreme case, we could compute the LM cost of all possible combinations at each hypernode, which will lead to an optimal heuristic. Of course this computation would be much more costly than the actual search using the cube growing algorithm. Due to efficiency considerations, we can not compute an optimal heuristic and thus the search will still be inexact.

7.2 Standard Heuristic (noLM Heuristic)

In this section we will analyze the heuristic proposed by Huang and Chiang (2007) in terms of translation quality and computational cost. In the next section we will introduce a new heuristic which maintains translation performance while reducing memory requirements at no computation time expense.

Huang and Chiang (2007) propose to compute an n-best list of translations without taking into account the LM scores, a so-called *noLM parse* (possibly taking into account recombination of hypotheses). Afterwards they compute the LM scores of these n-best derivations, and use these scores as the heuristic for the hyperedges involved in the derivations. The motivation behind this approach is that in the noLM pass we will compute a hopefully representative portion of the needed derivations, and thus the best of these scores should act as a heuristic for the hyperedge. However there is no guarantee that the explored space will be big enough. If when taking the LM into account we need the heuristic for a hyperedge which was not computed in the noLM pass, we just take the LM score of the first-best derivation for this hyperedge.

In the case of the noLM heuristic, we cannot guarantee its acceptability, as we cannot show that the hyperedges used in the noLM n-best computation will be reused in the parse including the language model information. In fact, the translations produced without language model differ much from the ones when the language model is taken into account, therefore the adequacy of this heuristic is not clear. The efficiency can be controlled by varying the size of the n-best list, however small values of n can increase the risk of inappropriate heuristic values, while too big values require too much computation time for obtaining the value of the heuristic.

7.3 Coarse Language Model Heuristic

In this section we will propose and analyze a new heuristic for the LM cost of the derivations that are still in the intermediate buffer. The main idea is to cluster the words in the target language into a reduced number of classes and to compute an optimistic LM score on these classes.

We first recall that, given an n-gram language model, the score of a word w given its context h (also called history) is given by the expression (Kneser and Ney, 1995)

$$p(w|h) = \begin{cases} \alpha(w|h) & \text{if } N(h,w) > 0 \\ \gamma(h)\alpha(w|\overline{h}) & \text{if } N(h,w) = 0 \end{cases} \tag{36}$$

where $N(h,w)$ corresponds to the word-history count in the training corpus, $\alpha(w|h)$ is the (discounted) relative frequency of the word-history pair, $\gamma(h)$ is a back-off weight, which also ensures a proper normalization of the probability distribution and $\overline{h}$ is a generalized history, that is, h with the last word dropped.

Now assume we have a mapping $\mathcal{M}$ from our target vocabulary $\mathcal{E}$ into a set of classes K, with $|K| \ll |\mathcal{E}|$

$$\begin{aligned} \mathcal{M} : \mathcal{E} &\to K \\ w &\mapsto \mathcal{M}_w \end{aligned} \tag{37}$$

We can extend the mapping to a sequence of words w_1^N just by concatenating the mappings of the individual words, i.e. $\mathcal{M}_{w_1^N} = \mathcal{M}_{w_1} \ldots \mathcal{M}_{w_N}$.

Given this mapping we now define our heuristic by taking the maximum LM probability associated with the words that get mapped to the same class. More formally, define the following functions corresponding to the quantities α and γ of Equation (36)

$$\alpha_\eta(w|h) = \max_{\substack{w':\mathcal{M}_{w'}=\mathcal{M}_w \\ h':\mathcal{M}_{h'}=\mathcal{M}_h}} \{\alpha(w'|h')\} \tag{38}$$

$$\gamma_\eta(h) = \max_{h':\mathcal{M}_{h'}=\mathcal{M}_h} \{\gamma(h')\} \tag{39}$$

and the resulting heuristic

$$\eta(w|h) = \begin{cases} \alpha_\eta(w|h) & \text{if } N(\mathcal{M}_w|\mathcal{M}_h) > 0 \\ \gamma_\eta(h)\alpha_\eta(w|\overline{h}) & \text{if } N(\mathcal{M}_w|\mathcal{M}_h) = 0 \end{cases} \tag{40}$$

The parameters of this heuristic function can be computed offline before the actual translation process and are stored in ARPA-format, like a normal LM. This allows the reuse of the existing code for handling language models.

Note that $\eta(w|h)$ does not define a probability distribution any more, as it is not normalized. This poses no problem, as we are looking for an upper bound of the language model probabilities. This set of upper bound values does not need to define a probability distribution itself.

This heuristic value is computed for the derivations as they are being produced, and it gets updated in the corresponding hyperedge. The motivation for this heuristic is that the expected similarity of the words which can be produced by the translation rules associated with a hyperedge and the contexts in this hyperedge can be captured with the given classes, and thus this optimistic language model score is able to predict future LM scores.

One could also think of a, at least at first glance, more straightforward approach. Given the mapping of words into classes, we could compute the mapping of the data used for training the language model, and then train a new language model on this data. This approach, however, has a big drawback for the usage as a heuristic. If a new language model is trained, the probabilities associated with it are in a completely different range, due to the reduced vocabulary size. Therefore the newly trained language model does not give enough information about the original language model.

Taking into account the derivations for which we compute the heuristic, we can consider that this heuristic in most of the cases is acceptable. This is because we take the maximum of every term involved in Equation (36). Note however that the conditions in the case distinction have changed. In particular, we move from testing the presence of a word-history pair to the presence of the corresponding classes. As the classes are more general than the words it can be the case that for some combination we use the event-seen case (first line in the case distinction of Equations (36) and (40) instead of the back-off case used when considering the words themselves. In practice, the probability of the event-seen case is expected to be higher, but we can not guarantee it.

Another source of discrepancy arises from the term $\gamma(h)$ (and the corresponding $\gamma_\eta(h)$) and unseen histories h. Again, it can happen that in considering $\mathcal{M}_h$ we shift from an unseen to a seen event. Depending on the definition of the γ function, this can have issues on the acceptability of the heuristic function. In our concrete case, we train our models using Kneser-Ney smoothing (Kneser and Ney, 1995) and use the SRI toolkit (Stolcke, 2002) for our implementation. Under this conditions, for unseen histories, $\gamma(h) = 1$ (or 0 in the negative log-probability space). That means that when C_h has been seen, our heuristic will

again not be acceptable. This, however, does not seem to have a big negative effect on the results. The generalization on other hypotheses along the same hyperedge can again not be guaranteed.

With respect to efficiency, this heuristic introduces a new language model into the translation process. However, the size of this language model is quite small, especially when compared with the full language model used in search, and thus the overhead of the additional LM computations is small. On the other side, when compared with the original heuristic, we eliminate the need of the noLM pass altogether.

There is still the open question of how to choose the word-to-class mapping $\mathcal{M}$. In this work we use automatically generated classes computed using the `mkcls` tool (Och, 1999) in a similar way as described by Martin et al (1995). It uses a maximum likelihood approach on a corpus by using a class bigram decomposition. This tool is widely used as part of the preprocessing steps when training statistical alignments using the GIZA++ tool (Och and Ney, 2003). This criterion seems to be adequate for our task, as both the words themselves and the context are taken into account.

Another possibility can be to use Part-of-Speech tags as word classes. This alternative is explored in (Vilar and Ney, 2009).

7.4 Comparison of the Heuristics

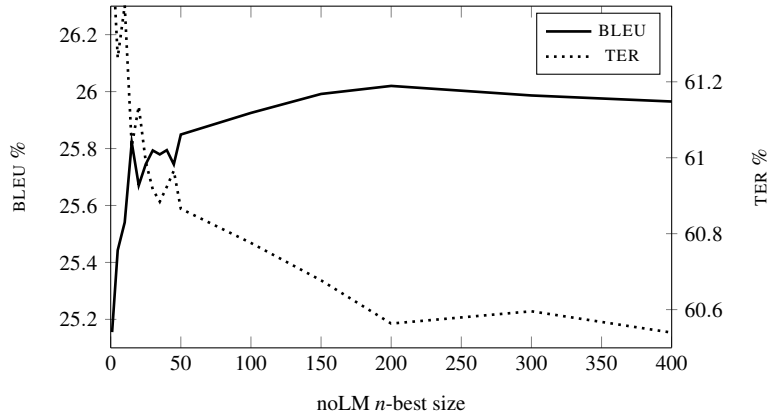
Figure 10 shows the results for the noLM heuristic[6]. The BLEU and TER scores are shown in Figure 10(a). The best results are achieved with a noLM n-best size of 200. The difference in performance is not too big and nearly optimal results can already be achieved with a noLM n-best size of 50. When looking into the computational resources, the difference becomes critical. Note that in this case we cannot simply compare the number of generated derivations as we did in Section 6, as the cost of computing the heuristic plays a critical role. We thus resort to memory and time measurements.

Figure 10(b) shows the memory usage dependent on the noLM n-best size. We can see that the memory requirements grow nearly linearly with the size of the n-best list (which is to be expected). The memory requirements using a noLM 50-best list is around 1.6GB. When using the 200-best list for optimal performance the memory requirements grow up to 6.5GB. For n-best sizes greater then 400, the memory requirements become prohibitive for the majority of current computers.

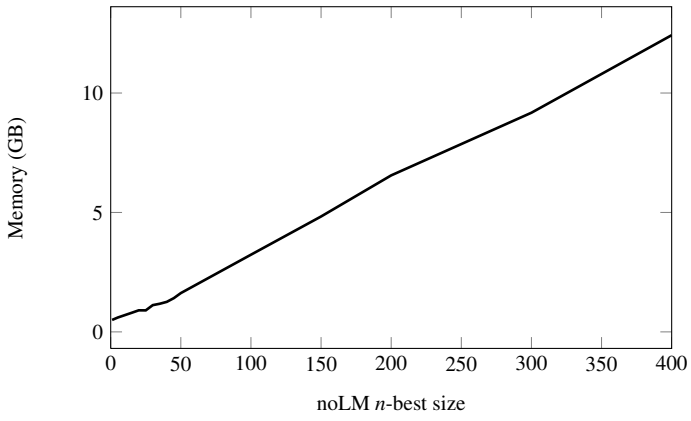
Computation time requirements are shown in Figure 10(c) as the average time needed for translating a sentence. The time requirements also grow with increasing noLM n-best size, but they stay quite reasonable, with a maximum of 6.5s per sentence. For optimum performance (200-best list), 5.2s per sentence are needed and for a 50-best heuristic, 4.3s. All time measures were taken on machines equipped with Quad-Core AMD Opteron processors with a clock speed of 2.2GHz.

The results for the coarse LM heuristic are show in Figure 11. It can be seen that the performance of the system using this heuristic is comparable with the noLM heuristic. It achieves a marginally better BLEU score at the cost of a marginally worse TER. The behaviour of this heuristic is somewhat more erratic than in noLM case. Memory requirements are shown in Figure 11(b). The memory requirements using the coarse LM heuristic are much lower than when using the noLM heuristic (note the different scale on the y-axis between Figures 10(b) and 11(b)), and they decrease as the number of classes increases.

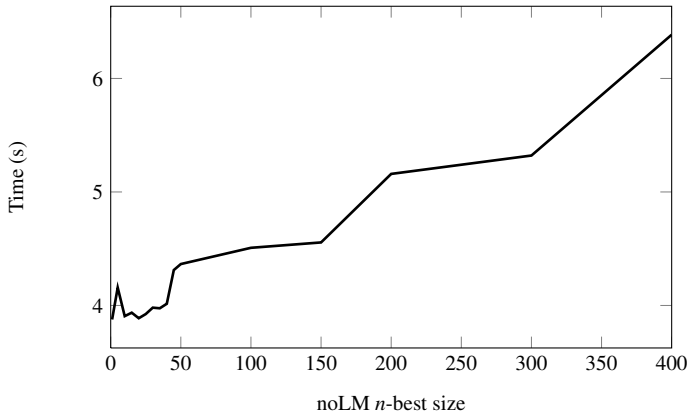
[6] Note that we used hypothesis recombination also in the noLM pass



(a) BLEU score

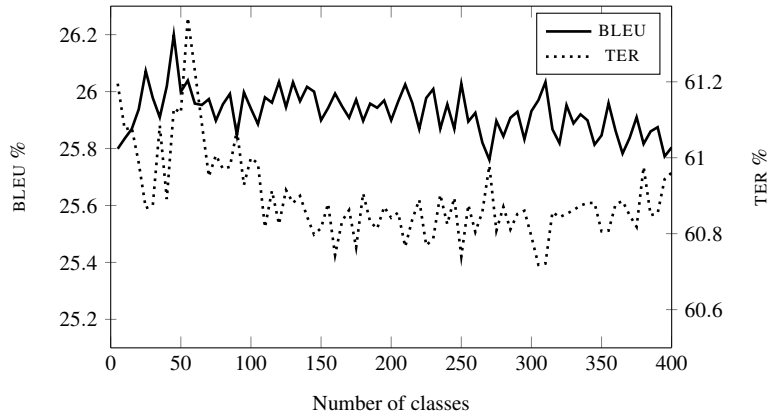


(b) Memory

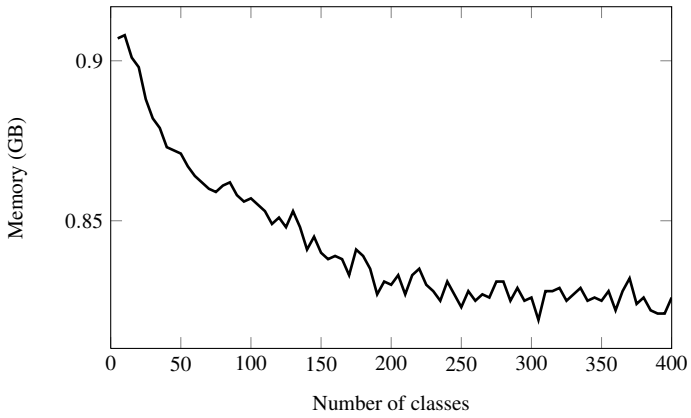


(c) Time per sentence

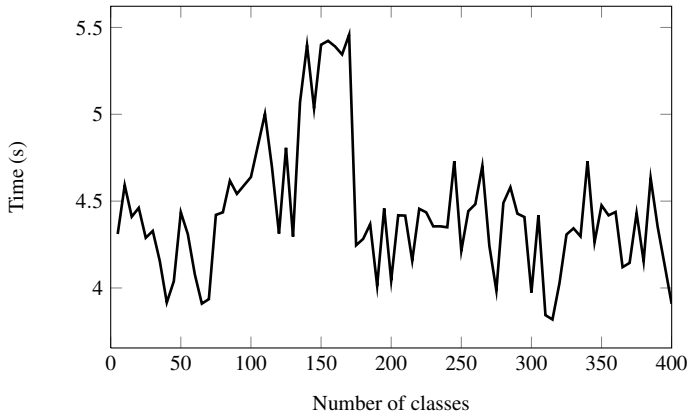
Fig. 10 Results using the noLM heuristic



(a) BLEU score



(b) Memory



(c) Time per sentence

Fig. 11 Results using the coarse LM heuristic

Table 1 Statistics for the Europarl German-English data. DEV corresponds to the WMT 2006 evaluation data, TEST to the 2008 evaluation data.

		German	English
TRAIN	Sentences	1 311 815	
	Running Words	34 398 651	36 090 085
	Vocabulary	336 347	118 112
	Singletons	168 686	47 507
DEV	Sentences	2 000	
	Running Words	55 118	58 761
	Vocabulary	9 211	6 549
	OOVs	284	77
	PPL	—	74.92
TEST	Sentences	2 000	
	Running Words	56 635	60 188
	Vocabulary	9 254	6 497
	OOVs	266	89
	PPL	—	85.21

Time requirements are shown in Figure 11(c) and are in general lower than for the case of noLM heuristics, except for very small values of n where the translation performance suffers severely. The time requirements also show an erratic behaviour. However, different workloads of the machines at experimentation time probably had a non-negligible effect on these measurements.

The behaviour of the noLM heuristic was expected. The increase in memory and time requirements is due to the increase effort in generating the noLM n-best lists. This does not imply an increase in translation quality, as the new hyperedges that are considered in the heuristic computation probably are not used in the actual translation process.

The coarse heuristic already achieves a good performance even for a small number of classes. This heuristic is able to simplify the LM computation scores and guide the parsing process in an efficient manner. This is consistent with the findings of Petrov et al (2008), albeit in a related but different context.

8 Experimental Results

In order to streamline the reading of the paper we have already included numerous intermediate experimental results as we described the different algorithms. In this section we will present a global comparison of all the algorithms discussed in this work.

The results in the preceding sections were computed on the German-English Europarl Task, as defined for the WMT evaluations (Callison-Burch et al, 2008). We used the 2006 evaluation data as development set for selecting the parameters of the models and to present the above results. With the outcome of those experiment we select the settings for each algorithm and present the final results on the blind data set composed of the evaluation data for the 2008 campaign. The statistics of the data can be seen in Table 1.

Figure 12 shows the performance of three main methods investigated in this work, depending on the average time needed for translating a sentence. In order to reduce the clutter of the graph, SCS cube pruning without coverage pruning was left out due to its similar behaviour to standard cube pruning. The coarse LM heuristic is also not included, as the

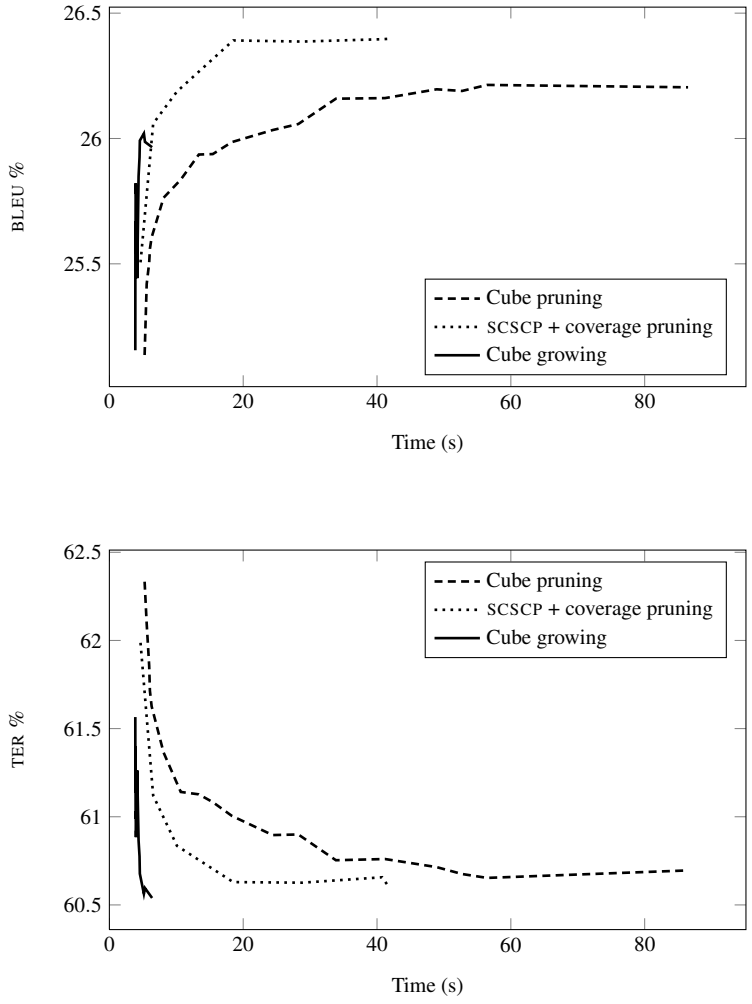


Fig. 12 Comparison of the efficiency of the three main search algorithms. SCSCP stands for source cardinality synchronous cube pruning.

parameters do not have a big impact on the performance of the algorithm. As can be seen in the graph, the cube pruning algorithm with coverage pruning is the algorithm that achieves the best BLEU score with a relatively low computation time. The cube growing algorithm is the best performing algorithm in terms of computation time, but this has a cost in translation quality. The performance in terms of TER is very similar for all three methods.

Depending on the task at hand we can choose one algorithm over the other. For tasks where speed is crucial, like for example interactive machine translation or online translation services, response time is far more critical that the slight gain in translation quality that can be achieved by applying a less aggressive pruning method.

Results on the blind test data, corresponding to the WMT 2008 evaluation data, can be found in Table 2. For computing these results we chose the best performing parameters tak-

Table 2 Results of the different search strategies on the 2008 WMT test data. Performance is measured in average translation time per sentence.

Search algorithm	BLEU [%]	TER [%]	Time [s]
Cube prune	27.2	60.6	78.3
Source cardinality synchronous CP	26.9	60.8	75.8
+ coverage pruning	27.1	60.9	20.2
Cube growing (noLM heuristic)	26.4	61.4	8.3
Cube growing (coarse LM heuristic)	26.7	61.0	7.0

ing into account the previous results. The performance of the three variants of cube pruning perform very similarly both in terms of BLEU and TER, the computation time is however much smaller when applying source cardinality synchronous search together with coverage pruning. The standard cube growing algorithm obtains somewhat worse results, but it performs much faster than the cube growing algorithms. Using the coarse LM heuristic the performance in terms of translation quality is comparable with the cube pruning methods. In terms of speed this is the best performing method.

In order to show that our system is competitive, Table 3 shows additional results achieved on this data. The first part of the table shows the two top performing systems in the 2008 evaluation campaign, both phrase-based systems using the Moses toolkit Koehn et al (2008); Déchelotte et al (2008). Note that the official results are rounded to the nearest integer value. It can be seen that the hierarchical system obtains very competitive results. Note also that no special adaptation for the task has been carried out, the results can be considered to be those of a baseline system. The table also shows additional results produced at our department. We provide two results for the phrase-based translation approach. The first one starts from the alignments produced by GIZA++ (Och and Ney, 2003) and can be seen as a baseline model. It can be seen that the hierarchical system performs better for this task. This result shows that the hierarchical approach is appropriate for the German-to-English translation task. The results corresponding to the best performing system available in our department are also reported, obtained using the so-called "forced alignments" training method described in (Wuebker et al, 2010). It is possible to adapt these method to the hierarchical phrase-based model, but such approaches are beyond the scope of this work.

Additionally, results obtained with the freely available Joshua toolkit Li et al (2009a) are included. The system was configured so that the search effort[7] was comparable to the cube prune setup reported in Table 2.

Current state-of-the-art systems are complex systems. Since we might not have been aware of the best setup for Joshua, we also include additional results on the WMT 2010 evaluation campaign on Table 4. John Hopkins University participated in this evaluation using Joshua, thus the system was trained by its original authors (Schwartz, 2010) and thus can be considered to be fully optimized. RWTH also participated with the open-source hierarchical phrase-based system Jane (Vilar et al, 2010) (among others), the same system used for the experimental results presented in this work. A detailed description of RWTH's submission can be found in Heger et al (2010). The scores are computed using the official euromatrix web interface for machine translation evaluation.[8]

[7] The phrase table was extracted using the tools provided in the Joshua toolkit. Using the phrase-table extracted for the other experiments made the memory requirements for Joshua prohibitive.

[8] `http://matrix.statmt.org/`

Table 3 Best available results on the 2008 WMT test data, in the official evaluation and at RWTH Aachen. The official results on BLEU are rounded to the nearest integer. No official TER scores were reported.

System	BLEU [%]	TER [%]	Time[s]
University of Edinburgh	28	—	—
Limsi	27	—	—
RWTH phrase-based (GIZA++ alignments)	26.3	60.9	29.3
RWTH phrase-based (forced alignments)	27.7	59.2	18.9
Joshua (RWTH run)	26.7	61.1	134.0

Table 4 Results for Jane and Joshua in the WMT 2010 evaluation campaign.

	Jane		Joshua	
	BLEU [%]	TER [%]	BLEU [%]	TER [%]
German-English	21.8	69.5	19.5	66.0
English-German	15.7	74.8	14.6	73.8
French-English	26.6	61.7	26.4	61.4
English-French	25.9	63.2	22.8	68.1

9 Conclusions

In this paper we have presented a detailed analysis of the hierarchical phrase-based translation approach, with special emphasis on efficient generation procedures. We have presented a novel extension to the cube pruning algorithm that increases the computational efficiency of this algorithm, reducing the average computation time per sentence to nearly one fourth. For this, we adapted methods from the source cardinality synchronous search organization widely used in standard phrase-bases translation. Cube pruning is nowadays considered the standard algorithm for the generation problem when using the hierarchical phrase-based translation model. The findings of this article may thus benefit a wide range of researchers.

We also investigated the cube growing algorithm, a reformulation of cube pruning with on-demand computation. We have studied the behaviour of the LM heuristic proposed, but not investigated in-depth, by the original authors. We have introduced a new heuristic that greatly reduces the memory consumption of the algorithm without penalty to running time or translation performance.

When comparing all the methods we found out that the cube growing algorithm performs worse than the cube pruning algorithm in terms of translation quality. The increased computational efficiency, however, may prove to be beneficial for applications where response time is crucial.

In addition we also provided an exact formal definition of the set of hierarchical phrases that can be extracted from a training corpus and formulated the translation problem using the dynamic programming formalism. This approach is more consistent with a wide range of literature on statistical machine translation.

References

Arun A, Dyer C, Haddow B, Blunsom P, Lopez A, Koehn P (2009) Monte Carlo Inference and Maximization for Phrase-based Translation. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009), Association for Computational Linguistics, Boulder, Colorado, pp 102–110, URL http://www.aclweb.org/anthology/W09-1114

Block HU (2000) Example-Based Incremental Synchronous Interpretation. In: Wahlster W (ed) Verbmobil: Foundations of Speech-to-Speech Translation, Springer Verlag, Berlin, Germany, pp 411–417

Callison-Burch C, Fordyce C, Koehn P, Monz C, Schroeder J (2008) Further Meta-Evaluation of Machine Translation. In: Proceedings of the Third Workshop on Statistical Machine Translation, Association for Computational Linguistics, Columbus, Ohio, USA, pp 70–106

Chappelier JC, Rajman M (1998) A Generalized CYK Algorithm for Parsing Stochastic CFG. In: Proceedings of the First Workshop on Tabulation in Parsing and Deduction, Paris, France, pp 133–137

Chiang D (2005) A Hierarchical Phrase-Based Model for Statistical Machine Translation. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Ann Arbor, Michigan, USA, pp 263–270

Chiang D (2007) Hierarchical Phrase-based Translation. Computational Linguistics 33(2):201–228

Cocke J (1969) Programming Languages and their Compilers: Preliminary Notes. Courant Institute of Mathematical Sciences, New York University

Déchelotte D, Adda G, Allauzen A, Bonneau-Maynard H, Galibert O, Gauvain JL, Langlais P, Yvon F (2008) Limsi's Statistical Translation Systems for WMT'08. In: Proceedings of the Third Workshop on Statistical Machine Translation, Association for Computational Linguistics, Columbus, Ohio, pp 107–110, URL http://www.aclweb.org/anthology/W/W08/W08-0310

Heger C, Wuebker J, Huck M, Leusch G, Mansour S, Stein D, Ney H (2010) The RWTH Aachen Machine Translation System for WMT 2010. In: Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR, Association for Computational Linguistics, Uppsala, Sweden, pp 93–97

Hopkins M, Langmead G (2009) Cube Pruning as Heuristic Search. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, pp 62–71, URL http://www.aclweb.org/anthology/D/D09/D09-1007

Huang L, Chiang D (2005) Better *k*-best Parsing. In: Proceedings of the 9th Internation Workshop on Parsing Technologies, pp 53–64

Huang L, Chiang D (2007) Forest Rescoring: Faster Decoding with Integrated Language Models. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, Czech Republic, pp 144–151

Iglesias G, de Gispert A, R Banga E, Byrne W (2009) Hierarchical Phrase-Based Translation with Weighted Finite State Transducers. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Boulder, Colorado, pp 433–441

Kasami T (1965) An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages. Tech. rep., Hawaii University Honolulu Department of Electrical Engineering

Kneser R, Ney H (1995) Improved Backing-off for M-gram Language Modeling. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processingw, Detroit, Michigan, USA, vol 1, pp 181–184

Koehn P (2003) Noun Phrase Translation,. PhD thesis, University of Southern California

Koehn P (2004) Pharaoh: a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models. In: Proceedings of the 6th Conference of the Association for Machine Translation in the Americas, Georgetown University, Washington DC, USA, pp 115–124

Koehn P, Hoang H, Birch A, Callison-Burch C, Federico M, Bertoldi N, Cowan B, Shen W, Moran C, Zens R, Dyer C, Bojar O, Constantin A, Herbst E (2007) Moses: Open Source Toolkit for Statistical Machine Translation. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL), Prague, Czech Republic, pp 177–180

Koehn P, Arun A, Hoang H (2008) Towards better Machine Translation Quality for the German-English Language Pairs. In: Proceedings of the Third Workshop on Statistical Machine Translation, Association for Computational Linguistics, Columbus, Ohio, pp 139–142, URL http://www.aclweb.org/anthology/W/W08/W08-0318

Li Z, Callison-Burch C, Dyer C, Khudanpur S, Schwartz L, Thornton W, Weese J, Zaidan O (2009a) Joshua: An Open Source Toolkit for Parsing-Based Machine Translation. In: Proceedings of the Workshop on Statistical Machine Translation, Athens, Greece, pp 135–139

Li Z, Eisner J, Khudanpur S (2009b) Variational Decoding for Statistical Machine Translation. 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, Association for Computational Linguistics, Suntec, Singapore, pp 593–601, URL http://www.aclweb.org/anthology/P/P09/P09-1067

Lopez A (2009) Translation as Weighted Deduction. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Athens, Greece, pp 532–540

Martin S, Liermann J, Ney H (1995) Algorithms for Bigram and Trigram Word Clustering. In: European Conference on Speech Communication and Technology, Madrid, Spain, pp 1253–1256

May J, Knight K (2006) Tiburon: A Weighted Tree Automata Toolkit. In: Proceedings of the Eleventh International Conference on Implementation and Application of Automata, pp 102–113

Och FJ (1999) An Efficient Method for Determining Bilingual Word Classes. In: Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics, Bergen, Norway, pp 8–12

Och FJ (2002) Statistical Machine Translation: From Single-Word Models to Alignment Templates. PhD thesis, RWTH Aachen University, Aachen, Germany

Och FJ (2003) Minimum Error Rate Training for Statistical Machine Translation. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, Japan, pp 160–167

Och FJ, Ney H (2003) A Systematic Comparison of Various Statistical Alignment Models. Computational Linguistics 29(1):19–51

Och FJ, Ney H (2004) The Alignment Template Approach to Statistical Machine Translation. Computational Linguistics 30(4):417–449

Papineni K, Roukos S, Ward T, Zhu WJ (2002) Bleu: a Method for Automatic Evaluation of Machine Translation. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pp 311–318

Petrov S, Haghighi A, Klein D (2008) Coarse-to-Fine Syntactic Machine Translation using Language Projections. In: Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, Honolulu, Hawaii, pp 108–116

Schwartz L (2010) Reproducible Results in Parsing-Based Machine Translation: The JHU Shared Task Submission. In: Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR, Association for Computational Linguistics, Uppsala, Sweden, pp 177–182

Snover M, Dorr B, Schwartz R, Micciulla L, Makhoul J (2006) A Study of Translation Edit Rate with Targeted Human Annotation. In: Proceedings of the 7th Conference of the Association for Machine Translation in the Americas, Cambridge, Massachusetts, USA, pp 223–231

Stolcke A (2002) SRILM – An Extensible Language Modeling Toolkit. In: Proceedings of the Seventh International Conference on Spoken Language Processing, ISCA, Denver, Colorado, USA, pp 901–904

Tillmann C, Ney H (2003) Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation. Computational Linguistics 29(1):97–133

Venugopal A, Zollmann A, Stephan V (2007) An Efficient Two-Pass Approach to Synchronous-CFG Driven Statistical MT. In: Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference, Association for Computational Linguistics, Rochester, New York, USA, pp 500–507

Vilar D, Ney H (2009) On LM Heuristics for the Cube Growing Algorithm. In: Proceedings of the Annual Conference of the European Association for Machine Translation (EAMT), Barcelona, Spain, pp 242–249

Vilar D, Stein D, Huck M, Ney H (2010) Jane: Open Source Hierarchical Translation, Extended with Reordering and Lexicon Models. In: Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR, Association for Computational Linguistics, Uppsala, Sweden, pp 262–270

Watanabe T, Tsukada H, Isozaki H (2006) Left-to-right Target Generation for Hierarchical Phrase-based Translation. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Sydney, Australia, pp 777–784

Wuebker J, Mauser A, Ney H (2010) Training Phrase Translation Models with Leaving-One-Out. In: 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, p to appear

Younger DH (1967) Recognition and Parsing of Context-Free Languages in Time n^3. Information and Control 2(10):189–208

Zens R (2002) Kontextabhängige Statistische Übersetzungsmodelle. Master's thesis, RWTH Aachen University

Zens R (2008) Phrase-based Statistical Machine Translation: Models, Search, Training. PhD thesis, RWTH Aachen University, Aachen, Germany

Zens R, Ney H (2008) Improvements in Dynamic Programming Beam Search for Phrase-based Statistical Machine Translation. In: International Workshop on Spoken Language Translation, Honolulu, Hawaii, pp 195–205

Zollmann A, Venugopal A (2006) Syntax Augmented Machine Translation Via Chart Parsing. In: Proceedings of the Workshop on Statistical Machine Translation, New York City, New York, USA, pp 138–141