Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS

Aleš Tamchyna

Feature Selection for Factored Phrase-Based Machine Translation

Institute of Formal and Applied Linguistics

Supervisor: RNDr. Ondřej Bojar, Ph.D.
Study program: Computer Science, Mathematical Linguistics

2012

I would like to thank my supervisor, RNDr. Ondřej Bojar, Ph.D., for his many valuable suggestions and for his readiness to provide helpful consults.

Název práce: Výběr rysů pro faktorový frázový strojový překlad
Autor: Aleš Tamchyna
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: RNDr. Ondřej Bojar, Ph.D.
e-mail vedoucího: Ondrej.Bojar@mff.cuni.cz

Abstrakt: V předkládané práci zkoumáme problematiku faktorových modelů ve strojovém překladu. Podáváme důkladný teoretický popis této oblasti strojového překladu. Popisujeme metodu sloužící ke zkoumání složitosti faktorových modelů a ověřujeme její fungování v praxi. Představujeme softwarový nástroj pro automatické vytváření experimentů strojového překladu a pro prohledávání prostoru možných konfigurací. V experimentální části pak ověřujeme svoje analýzy a podáváme obraz možností faktorových systémů. Naznačujeme směry, kterými lze dosáhnout zlepšení kvality strojového překladu, avšak shledáváme, že tyto možnosti nelze zkoumat zcela automaticky.

Klíčová slova: strojový překlad, faktorové modely, výběr rysů

Title: Feature Selection for Factored Phrase-Based Machine Translation
Author: Aleš Tamchyna
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Ondřej Bojar, Ph.D.
Supervisor's e-mail address: Ondrej.Bojar@mff.cuni.cz

Abstract: In the presented work we investigate factored models for machine translation. We provide a thorough theoretical description of this machine translation paradigm. We describe a method for evaluating the complexity of factored models and verify its usefulness in practice. We present a software tool for automatic creation of machine translation experiments and search in the space of possible configurations. In the experimental part of the work we verify our analyses and give some insight into the potential of factored systems. We indicate some of the possible directions that lead to improvement in translation quality, however we conclude that it is not possible to explore these options in a fully automatic way.

Keywords: machine translation, factored models, feature selection

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Research in machine translation started with an attempt to capture linguistic rules that control the transfer of meaning from one language to another. Originally, systems for machine translation were therefore rule-based—for each language, experts would develop dictionaries and large sets of rules that facilitated the transfer. Creating such a system is very time-consuming and costly, yet while the final product can be practically useful, it lacks the ability to generalize beyond the incorporated set of rules. For example, when an unexpected grammatical construction appears in the input, these systems will often fall back to mere word-by-word translation. Because of the cost of building such a system, many of them are used commercially even today. Some companies invested decades of development into refining the dictionaries and rules, which can now capture most grammatical phenomena and many exceptions in the particular languages.

With the increasing computational power, a statistical approach to machine translation became possible. Using statistics to model translation was suggested in 1949 by Warren Weaver, but a practical model was developed around 1990. Word-based translation (described in Section 2.1.2) became a strong paradigm and today, it forms the basis for the most widespread approach to machine translation, namely phrase-based machine translation.

In this paradigm, parallel training data are used for estimating probabilities of phrase translations. When a new sentence is to be translated, it is decomposed into phrases and the decoder searches for the most probable combination of phrase translations, which then constitute the translation of the whole sentence. We describe training and decoding of phrase-based models thoroughly in Chapter 2. Decoding in the context of machine translation means the search for the most probable hypothesis in the space of possible translations.

1.1 Motivation

Phrase-based models use no linguistic information for translating. They see the translated phrases as mere strings of tokens; if we substituted each word from the vocabulary with a numerical identifier and trained a translation system on this modified data, we would obtain identical results.

This fact is not necessarily a weakness. On average, phrase-based methods outperform most other approaches to machine translation. They translate idiomatic expressions easily (if they are observed during training), they have very strong models for local context and are robust to errors and noise in the data. Moreover, the fact that statistical models are language-independent makes it possible to build large-scale multilingual translation systems in a very short amount of time—such a task would have been impossible with rule-based systems.

But still, phrase-based models suffer from inherent limitations that some linguistic insight might help to overcome. Different kinds of problems arise for various types of languages, however we encounter many of the fundamental issues of phrase-based translation if we translate from English (a language with fixed word order and poor morphology) into Czech.

In Czech, most lexemes have a number of surface forms, due to the morphological richness of the language. Phrase-based translation has no notion of morphology, so each form is modeled completely independently of the others. A phrase-based system is also unable to output a form not seen in the training data (and even very large corpora do not contain all forms of each word).

Czech has another property that can make the phrase-based modeling of translation difficult. The relatively free word order can potentially break the notion that language can be modeled by n-grams (short sequences of words).

Long-distance dependencies present another challenge for phrase-based models. This problem is closely related to the free word order and rich morphology; surface forms of words can depend on constituents from distant parts of the sentence[1]. Phrase-based models only capture local context of a fixed size; the context is not necessarily short, but it is always possible to "pump" words between such constituents to exceed the scope. Also, as we move the words further apart, the data to model the context become more sparse and the statistics lose reliability.

In this thesis, we explore the paradigm of factored translation models. These models aspire to incorporate some linguistic information into phrase-based translation. Words are no longer viewed as indivisible units, but rather as vectors of factors which describe various linguistic properties of the words. We can view this as a decomposition of the translation problem into a series of modeling steps.

We can illustrate this shift using the machine translation pyramid (Figure 1.1), where the direct transfer represents a phrase-based model and the bold lines show a system with more linguistic insight. First, analysis of the source side is performed. This enables the system to generalize beyond surface forms, to model morphology or syntax. The transfer is then modeled on this more general level. Finally, the translated information is used on the target side to generate the surface form of the translation. Factored models attempt to include, to a certain degree, both the analysis and the generation, while staying in the simple phrase-based translation paradigm.

Theoretically, factored models have the potential to alleviate most of the problems mentioned above. We can, for example, translate lemmas instead of

[1]The dependencies can go beyond the scope of a sentence, but such context is only slowly starting to be considered in machine translation.

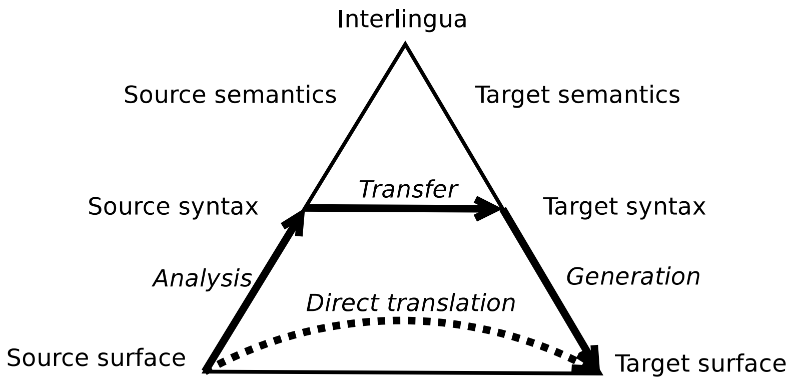


Figure 1.1: Machine translation pyramid. The dotted line shows surface translation, such as word-based or phrase-based. The bold lines show a more linguistically informed system.

forms (making it possible to obtain more robust models) and generate the forms using lemma and morphological tag on the target side. This way, we could reduce the negative impact of target-side rich morphology.

By using information from the dependency structure as factors (e.g. the analytical function of the parent word), we could incorporate a (very shallow) notion of syntax into our model, possibly enabling them to capture (to a certain extent) the free word order and long-distance dependencies.

Overall, the decomposition into factors could enable us to build more general, more robust models of specific linguistic phenomena. These models provide the decoder with better statistics, potentially leading to improvements in translation quality.

1.2 Goals

Factored models have been used since their introduction in 2007 by many researches. The paradigm is therefore quite well explored. In this work, we attempt to create a method for automatic search in possible factored configurations.

Several tasks need to be solved for this search to be possible. We need a way of enumerating the possible factored setups. A reliable evaluation criterion is necessary to guide the search in this space. As machine translation experiments can be very time-consuming, we also need to develop a heuristic algorithm for estimating the complexity of factored configurations before running the experiments. Factored models depend on rich linguistic annotation. Part of this work will therefore focus on describing the possible features and their contribution to translation quality. We provide experimental evidence to support our claims and arguments throughout this work.

1.3 Outline

We discuss the theory involved in phrase-based machine translation in Chapter 2. In Chapter 3 we focus on factored models and we set our work in the context of previous research.

Chapter 4 contains our analysis of factored models in terms of complexity. This chapter also discusses the factors used in our experiments. We present our tool for automatic creation of machine translation experiments in this chapter as well. We analyze the options for evaluating factored setups and discuss the number of possible factored systems that need to be explored if we aim at a fully automatic search for system configurations.

In Chapter 5 we provide an analysis of the impact of a few parameter settings for Moses, but the focus on this chapter is in the experimental evaluation of several factored scenarios.

The final Chapter 6 contains a discussion on our findings. We conclude by reiterating the goals of this work and provide some directions for possible future research in this area.

Chapter 2

Phrase-Based Translation

2.1 Introduction

Statistical phrase-based machine translation (PBMT, Koehn et al. 2003) is probably the most popular method today. It is relatively simple, but more sophisticated models have so far been unsuccessful in convincingly surpassing its performance across various language pairs. PBMT is therefore widely deployed in practice to provide general translation for public, such as Google Translate[1], as well as to aid professional translators in specialized companies.

In this chapter, we will describe the theoretical background of phrase-based machine translation.

2.1.1 The Noisy Channel Model

The basic idea behind statistical machine translation comes from information theory. It uses the *noisy channel model* that was originally developed for modeling data transfer over an unreliable medium, which scrambles the information. In this view, translation is in fact *decoding* from scrambled English (i.e., the foreign language) into English. The noisy channel model uses the Bayes' theorem to decompose the problem into two models as follows:

$$P(e|f) = \frac{P(f|e)P(e)}{P(f)} \tag{2.1}$$

In the equation, e denotes the English sentence (i.e. the translation we are looking for) and f the input (foreign or "French") sentence. The denominator remains constant during decoding, so it can be disregarded. The goal of decoding is then to find an English sentence that maximizes the right-hand side of the equation, i.e. to find an e that is at the same time a probable translation of f and a probable English sentence.

[1] `http://translate.google.com/`

Let E be the set of all English sentences, then, formally we are trying to find $\hat{e}$ that satisfies:

$$\hat{e} = \operatorname*{argmax}_{e \in E} P(f|e)P(e) \tag{2.2}$$

We are actually modeling the opposite translation direction—in the approaches used today, both directions are modeled and the probabilities are combined. In the following sections, we will adhere to this reversed notation merely for consistency.

Each of the probability distributions can then be modeled separately. We call the model of $P(f|e)$ the *translation model* and $P(e)$ the *language model.* We will now shortly describe a key approach to modeling the translation probability. The language model is discussed in Section 2.5.

2.1.2 IBM Models

Phrase-based models build upon word-based translation, specifically the so-called IBM Models (Brown et al., 1993). These models are used (with some modifications) to compute the word alignment on parallel training data which is then used to extract phrases for phrase-based translation. Apart from the original work, the description in this section is also based on (Koehn, 2010).

IBM Models are a series of increasingly complex models for machine translation—the first model only estimates lexical translation probabilities, successive models then add estimation of reordering or word fertility. The final parameters of each model serve as the initial setting for the following model. For all models, the expectation maximization (EM) algorithm is used to estimate their parameters.

IBM Model 1 simply estimates lexical translation probabilities, without regard for reordering or the number of words aligned to a single token. Formally, the probability of sentence $\mathbf{f} = (f_1, \ldots, f_m)$ given $\mathbf{e} = (e_1, \ldots, e_l)$ is:

$$P(\mathbf{f}|\mathbf{e}) = \frac{\epsilon}{(l+1)^m} \sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} \prod_{j=1}^{m} t(f_j|e_{a_j}) \tag{2.3}$$

The constant ϵ is required for correct normalization of probability. The English sentence is preceded by an added NULL token, to which additionally generated foreign words are aligned. For this reason, there is $l+1$ in the denominator. The $t(f|e)$ represent lexical translation probability of word f given e. The various sums represent alignment points for each foreign word. It would be impractical to iterate over all possible alignments—however, thanks to the simplicity of the model, the equation is equivalent to:

$$P(\mathbf{f}|\mathbf{e}) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j|e_i) \tag{2.4}$$

The lexical translation probabilities are estimated using EM algorithm—this is the first model, so initially, all translations are considered equally likely. By iteratively applying the current model to the data and re-estimating its parameters, the algorithm converges to a global optimum.

IBM Model 2 adds an absolute model for alignment. Translation can be thought of as divided into 2 steps, the lexical translation and the alignment step. It can be expressed formally as follows.

$$P(\mathbf{f}|\mathbf{e}) = \epsilon \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j|e_i) a(i|j,m,l) \tag{2.5}$$

The main difference is the addition of $a(i|j,m,l)$, i.e. the probability of j-th foreign word to be aligned to i-th English word given the length of foreign and English sentences. Both t and a are estimated jointly from the data using EM until convergence to a global optimum, similarly as in IBM Model 1.

IBM Model 3 adds two important components:

- Word fertility.
- NULL token insertion.

An English word may be translated to one, two or more foreign words. This is captured by the model of fertility. Since the fertility of the special NULL token depends on the sentence length, the probability of inserting it is modeled separately. The translation now has four steps—fertility, NULL insertion, lexical translation and alignment. The last 2 steps are modeled similarly as in the previous models.

The mathematical formulation of IBM Model 3 is quite complicated and beyond the scope of this overview. This model is however important in one more aspect—the simplification which cancelled out the exponential number of possible alignments is no longer possible and running full EM therefore becomes unfeasible. The space of possible alignments has to be sampled. The sampling is done via hill climbing from the most probable alignment (the *Viterbi* alignment) according to IBM Model 2. In each iteration of the hill climbing, we generate all neighbors of the current best alignment and calculate their probability according to IBM Model 3. A neighbor is an alignment that differs in either a move (i.e. one word is aligned differently) or a swap (alignment of two words is switched).

To reduce the risk of finding a low local optimum, *pegged* alignments are also searched. A pegged alignment has one alignment point (i,j) fixed throughout the hill-climbing procedure. The sample of alignment space contains the best alignments with (i,j) pegged for all i and j.

IBM Model 4 extends the previous model in two ways—by explicitly modelling relative alignment and by incorporating word classes. The alignment of each word is now conditioned on the alignment of its predecessor. This helps to keep coherent alignments. Word classes (both on the foreign and the English side) are a way to generalize beyond surface word forms and to reduce data sparsity. They are usually found by unsupervised clustering of the data.

Model 5 fixes a deficiency of the previous models. In models 3 and 4 it is possible to generate non-strings by assigning the same position to multiple words. That means that much of the probability mass is wasted on impossible solutions. Model 5 amends the deficiency by keeping track of vacant word positions and only allowing words to be placed in these positions.

2.2 Overview of Training

By training a translation system, we mean the whole process—data pre-processing, model estimation, tuning etc. From a machine-learning perspective, the tuning phase would be regarded as the actual training (where we present the system with labelled training examples: sentences and their BLEU scores).

To train a phrase-based machine translation (MT) system, a parallel corpus is a necessary data source. This is a sentence-aligned bilingual corpus. These corpora are usually extracted from naturally parallel sources such as international law documents, fiction, movie subtitles or multi-lingual web pages. Since the sentence correspondence is not necessarily one-to-one, tools for sentence alignment are applied, e.g. Hunalign (Varga et al., 2005).

During training, phrases and their translations are extracted from the parallel corpus and their probability is modeled by maximum likelihood estimation (MLE). In order to extract phrases, we need phrase alignment. Several techniques for extracting aligned phrases directly from the corpus have been suggested, but in practice, a heuristic approach that derives phrase pairs from word alignment is the most successful.

After phrase extraction and estimating their probabilities (translation probabilities in both directions and lexical weights), we have the translation model.

A language model is trained on a set of target-side monolingual data to predict which hypotheses are more probable sentences in the output language.

These two models are combined as features into a log-linear model. Weights of the features are found empirically by optimizing output translation towards a goal function that gauges translation quality (i.e. a machine translation *metric*[2]). Both the optimization procedure and metrics are a very active area of research. The most commonly used optimization algorithm in the Minimum Error Rate Training (MERT, Och 2003, see Section 2.7.1). The goal function is usually BLEU (Papineni et al., 2002).

2.3 Word Alignment

The IBM Models became the state-of-the-art for machine translation at their time. Today they are rarely used in practice for decoding. However, they are still applied to solving an important subproblem of training a phrase-based MT system, namely word alignment.

[2]In fact, most practically used functions do not have the mathematical properties of a metric, so the term "measure" would be more appropriate.

It was found that for word alignment, it is beneficial to replace the IBM Model 2 with an HMM alignment model (Vogel et al., 1996) which produces more coherent alignments and therefore a better starting point for IBM Model 3 hill climbing.

Word alignment is acquired by running several iterations of each IBM model and computing the Viterbi alignment of the training data.

2.3.1 Alignment Symmetrization

Because word alignment is a function, i.e. each foreign word can be traced to at most one English word, we cannot model m-to-n mappings, which are very common. To overcome this limitation, word alignment is computed both ways, $e \rightarrow f$ and $f \rightarrow e$, and symmetrized. Note that the noisy channel interpretation of machine translation begins to fade at this point.

There are various heuristics for symmetrization. The very basic two approaches are *union* and *intersection.* Aiming at highest possible recall, the union takes all alignment points[3] from both directions. The intersection outputs only the points present in both directions and has high precision.

An improved approach to symmetrization was first suggested in Och and Ney (2003). The terminology we will follow is defined in Koehn et al. (2005).

All heuristics begin with the alignment intersection and extend it. First, only points which are in the union and neighbor an existing alignment point are added. Depending on the definition of "neighboring", the heuristics is either *grow* (only points directly to the left, right, above or below) or *grow-diag* (also include diagonal neighbors). This symmetrization can be further extended by the *final* or *final-and* algorithm. The first algorithm adds all points from the union from which at least one of the words is still unaligned. The *final-and* is more restrictive—it only adds points where both words are unaligned. The commonly used combinations are *grow-diag-final* and *grow-diag-final-and.*

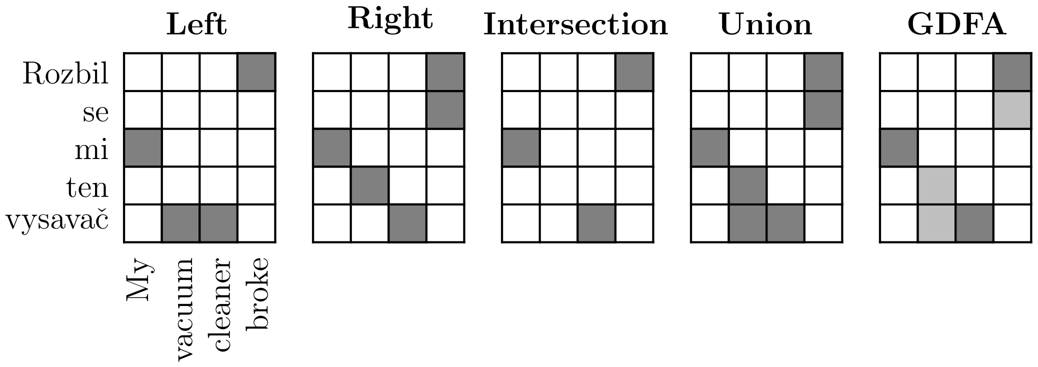


Figure 2.1: Comparison of alignment symmetrizations.

Figure 2.1 shows the symmetrizations on a sample sentence. The *left* direction

[3]Alignment points are pairs $(i, j); i = (1, \ldots, l_e), j = (1, \ldots, l_f)$

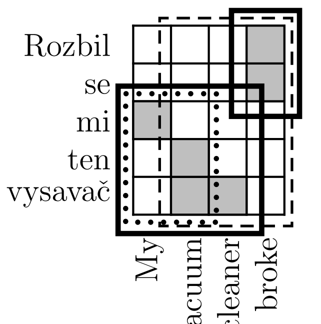
does not align the Czech clitic “se” or the demonstrative pronoun “ten” because no English word can be aligned to more than one word. The word “vysavač” is correctly aligned to “vacuum cleaner”—alignment is not necessarily an injective function. On the other hand, the *right* alignment cannot align the “vacuum cleaner”, but the clitic is aligned correctly to the verb “broke”. In this case, *GDFA* is the same as the *union*. In fact, all available alignment points were added in the *grow-diag* step (denoted by light gray color).

2.4 Phrase Extraction

Phrases are contiguous sequences of words. They do not necessarily correspond to the linguistic notion—even sequences like “not go to” or “.” can be phrases (if they are seen in the training data).

Symmetrized word alignment is used to heuristically extract phrases. Given a word-aligned sentence pair from the training data, we simply go over all possible phrase pairs (phrases have a defined maximum length, e.g. 7) and check whether the pair is consistent with the symmetrized word alignment. If so, we output it.

A phrase is consistent if all its words are aligned within the phrase and not outside of it. Figure 2.2 shows the notion of consistency on the word alignment matrix—if we are extracting a phrase that spans over columns i to $i+l_i$ and over rows j to $j+l_j$, then the columns must not contain a point outside the rows (smaller than j or larger than $j+l_j$) and vice versa.



mi — My
mi ten vysavač — My vacuum cleaner
Rozbil se mi ten vysavač — My vacuum cleaner broke
ten vysavač — vacuum cleaner

Figure 2.2: Examples of extracted phrases. Dotted/dashed rectangles are inconsistent phrases, solid-line rectangles represent the consistent ones.

The phrases are usually quite redundant, many overlapping phrases which are consistent with the word alignment can normally be extracted. This fact makes the model robust towards unexpected input data. We can however retain the robustness even if the phrase table is filtered to a fraction of its size. Methods for reducing phrase table size without loss of translation quality have been successfully developed (see e.g. Johnson et al., 2007).

After extracting all phrases from the training data, the set of phrases is sorted and phrase scores are computed. In Figure 2.3, the total number of occurrences of the phrase "estimated in the programme" is 9. The probability $P($"naznačena v programu"$|$"estimated in the programme"$)$ equals $3/9 = 1/3$. Thanks to the sorting, this can be computed easily and the scoring function does not need to cache large amounts data. The *inverse* phrase table (i.e. with swapped source and target language) is also sorted and scored, leading to an estimate of $P(e|f)$.

estimated in the programme	naznačena v programu
estimated in the programme	naznačena v programu
estimated in the programme	naznačena v programu
estimated in the programme	odhadován v programu
estimated in the programme	odhadovány v programu
estimated in the programme	odhadovány v programu
estimated in the programme	předpokládal program
estimated in the programme	v programu uvedeným
estimated in the programme	v programu uvedeným

Figure 2.3: Excerpt from a sorted phrase table.

Additionally, lexical weights are calculated for both directions. They are a method for smoothing the phrase table. Infrequent phrases have unreliable probability estimates; for instance many long phrases occur together only once in the corpus, resulting in $P(\mathbf{e}|\mathbf{f}) = P(\mathbf{f}|\mathbf{e}) = 1$. Several methods exist for computing lexical weights. The most common one is based on word alignment inside the phrase (Koehn, 2003). The probability of each *foreign* word f_j is estimated as the average of lexical translation probabilities $w(f_j, e_i)$ over the English words aligned to it. Thus for the phrase $(\mathbf{e}, \mathbf{f})$ with the set of alignment points a, the lexical weight is:

$$\text{lex}(\mathbf{f}|\mathbf{e}, a) = \prod_{j=1}^{l_f} \frac{1}{|i|(i,j) \in a|} \sum_{\forall(i,j)\in a} w(f_j, e_i) \qquad (2.6)$$

2.5 Language Model

The task of language modeling in machine translation is to estimate how likely a sequence of English words $\mathbf{w} = (w_1, \ldots, w_l)$ is correct English[4].

When translating, the decoder generates translation hypotheses which are probable according to the translation model (i.e. the phrase table). The language model then scores these hypotheses according to how probable (common, fluent) they are in English. The final translation is then a compromise—the sentence that is both fluent and a good translation of the input.

Similarly to the translation model, sequence probabilities are learned from data using maximum likelihood estimation. For language modeling, only mono-

[4]We use English to denote the target language, similarly as we did in the description of the noisy channel model.

lingual data are needed (a resource available in much larger amounts than parallel texts).

Naturally, the prediction of the whole sequence $\mathbf{e}$ has to be decomposed, so that it can be reliably estimated. The most common approach are *n-gram* language models which build upon the Markov assumption: a word depends only on a limited, fixed number of preceding words. The decomposition is done as follows:

$$\begin{aligned} P(\mathbf{w}) &= P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\dots P(w_l|w_1,\dots,w_{l-1}) \\ &\approx P(w_1)P(w_2|w_1)\dots P(w_l|w_{l-n},\dots,w_{l-1}) \end{aligned}$$

The first equality follows from the chain rule and the second from n-th order Markov assumption. Each word is then modeled by at most n preceding words and the probability of the whole sequence is the product of probabilities of individual words.

This decomposition is still not sufficient. We have no guarantee that the training data includes all possible bigrams or trigrams (and most of the seen n-grams will be too rare to be estimated reliably). Furthermore, today's machine translation systems commonly use even 5-grams or 6-grams in language models.

A detailed description of techniques applied to this problem is beyond the scope of this work (language modeling is an independent task with a wide range of applications for natural language processing). In machine translation, the most common approach is a modified version of Kneser-Ney smoothing (Chen and Goodman, 1999). The following equation is used to compute the probability of each word:

$$\begin{aligned} P(w_i|w_{i-n},\dots,w_{i-1}) =&\alpha(w_i|w_{i-n},\dots,w_{i-1}) \\ &+ \gamma(w_{i-n},\dots,w_{i-1})P_I(w_i|w_{i-n+1},\dots,w_{i-1}) \end{aligned}$$

The raw probability as estimated from data by MLE is expressed by α, but for each seen n-gram, absolute discounting of the number of occurrences is used. A different discount value is used for n-grams that occur once, twice or three or more times. The discount value is based on counts of these n-grams in the data.

Note that the back-off is applied even when the whole n-gram was observed in the data. This interpolation leads to a more reliable estimation of rare high-order n-grams. The γ function serves as the interpolation weight. It assigns a certain amount of probability mass to unseen words that follow $w_{i-n},\dots,w_{i-1}$.

The back-off is recursive—the probability of a trigram is calculated as interpolation between the trigram MLE and bigram back-off, which in turn is interpolated from bigram MLE and back-off to unigrams.

2.6 Log-Linear Model

In phrase-based machine translation, log-linear model is used to score translation hypotheses. The model allows to decompose the translation score into an arbitrary set of features. Each feature has a weight which determines its importance. The score of hypothesis x is:

$$P(x) = \exp \sum_{i=1}^{n} \lambda_i h_i(x) \tag{2.7}$$

The basic features h_i in phrase-based machine translation are:

- phrase translation probabilities $P(\mathbf{f}|\mathbf{e})$ and $P(\mathbf{e}|\mathbf{f})$,
- phrase lexical weights $\text{lex}(\mathbf{f}|\mathbf{e})$ and $\text{lex}(\mathbf{e}|\mathbf{f})$,
- phrase penalty,
- language model score of $\mathbf{e}$,
- word penalty w,
- distortion penalty d.

Most of the features have been introduced in previous sections. Phrase penalty is 1 for each phrase in the output. Similarly, for each output word, the word penalty is 1.

Distortion penalty is very simple as well[5]. In the Moses decoder, distortion penalty is defined this way: for each phrase, its value is the distance (measured in words) between its beginning and the end of the preceding phrase. This *distance-based* reordering can be replaced by more sophisticated models, such as lexicalized reordering.

2.7 Tuning Model Parameters

We have described how the features h_i are calculated, but so far we omitted the estimation of weights λ_i. In log-linear models, the standard way is to find weights that maximize mutual information between foreign and English sentences in the set S:

$$\hat{\lambda}_1^n = \operatorname*{argmax}_{\lambda_1^n} \sum_{s \in S} log P_{\lambda_1^n}(\mathbf{f_s}|\mathbf{e_s}) \tag{2.8}$$

[5]Note that in word-based models, a great deal of computation went into modeling word reordering. Much of this information is included implicitly in the extracted phrases.

Och (2003) argued that optimization should instead be done toward a metric of translation quality, such as BLEU[6] (see Section 2.9), on a development (or *tuning*) set of sentences. Today, model parameters are almost always tuned this way. In this section, we will describe a few methods developed for this task.

2.7.1 Minimum Error Rate Training

Minimum Error Rate Training (MERT) was developed in the same work (Och, 2003) and has become a de-facto standard algorithm for tuning. The tuning process is iterative:

1. Set all weights to some initial values.
2. Translate the tuning set using the current weights; for each sentence, output n best translations and their feature scores.
3. Run one iteration of MERT to get a new set of weights.
4. If the n-best lists are identical to the previous iteration, return the current weights and exit. Else go back to 2.

The input for MERT is a set of *n-best lists*—the n best translations for each sentence in the tuning set. A vector of feature scores is associated with each sentence.

First, each translation is scored by the objective function (such as BLEU). In each n-best list, the sentence with the best score is assumed to be the best translation. The goal of MERT then is to find a set of weights that will maximize the overall score, i.e. move good translations to the top of the n-best lists.

Note that there is no guarantee that with the new set of weights, the same translations will appear in the n-best list. The previous best translation may not be in it; a whole new set of candidate translations might constitute the n-best list instead. By running the optimization iteratively on new n-best lists (these lists are usually combined with n-best lists from previous runs to make the optimization more stable), we can achieve ever higher quality of translations, until the weights converge or the number of iterations is too high. MERT is not guaranteed to find a global optimum and not even to converge.

MERT addresses the dimensionality of the weight space (effectively $\mathbb{R}^n$ for n weights) by optimizing each weight separately. The line search can be further simplified—very small changes in the weight value are unlikely to make a difference. In fact, the only changes that affect the value of the objective function are those that move some candidate translation to the top of an n-best list. For each sentence, these "breaking points" can be identified in time linear with respect to the n-best list size. Combining points from all n-best lists (i.e. of n best

[6] Various metrics have been proposed for tuning. However Cer et al. (2010b) have shown that despite the inherent limitations, BLEU is still the most appropriate metric for tuning the model parameters.

translations of all sentences in the tuning set) leads to a list of intervals. The optimization of one weight is thus reduced to finding the interval with the highest score.

While the line search is globally optimal (in the one dimension), overall, the procedure is likely to reach a local optimum. MERT is therefore usually run from a number of different starting positions and the best set of weights is used.

After convergence (or reaching a pre-set maximum number of iterations), the weights for log-linear model are known and the system training is finished.

Despite its widespread usage, MERT has quite serious limitations. When the log-linear model contains a large number of features (common in more complex factored scenarios), MERT tends to become quite unstable, sometimes to the point where it diverges. Moreover, even among successful (converged) runs, the deviation of translation quality can be quite high—results differing by as much as 1 absolute BLEU point are not uncommon. This can have impairing effects on experiments, as some expected improvement might be due just to random MERT variation, or vice versa—a hopeful technique could be discarded because of "bad luck" during optimization.

Approaches to mitigating this problem have been proposed. This simplest is running MERT several times and taking the average, observing carefully the standard deviation. A more sophisticated method is introduced in Clark et al. (2011). Along with the paper, the authors developed an implementation which samples multiple MERT runs and performs tests of statistical significance. This technique is more fine-grained and can therefore distinguish between two systems even when their final scores do not seem to be significantly different.

2.7.2 Pairwise Ranking Optimization

PRO (Hopkins and May, 2011) is a new technique developed as a replacement of MERT. It is advertised to be stable and to scale to large numbers of features.

PRO has a similar architecture in the sense that its input are n-best lists along with feature scores and that the procedure is run iteratively. The main difference is that PRO attempts to optimize the ranking inside the n-best lists—it should be as similar as possible to ranking according to the objective function (e.g. BLEU). MERT, on the other hand, focuses only on the hypothesis on top of each n-best list, disregarding the rest.

In each iteration, pairs of hypotheses are sampled from the n-best lists (exhaustive extraction of all sentence pairs is computationally prohibitive). Sampling is done with focus on the margin between hypothesis scores—pairs with large difference in scores are preferred. Ranking is then converted to a binary classification task. Let x_1 and x_2 be feature vectors of a sampled sentence pair and let the score of x_1 be higher than x_2. Then the following training instances are added: $(x_1 - x_2, +)$, $(x_2 - x_1, -)$.

The training data are given to a binary classifier. Its weight vector after training is then used as the new set of weights. Because of the random sampling, the n-best lists cannot be expected to converge. This algorithm is thus simply run for a pre-defined number of iterations. Afterwards, the set of weights which lead to the highest score (by the objective function) is output.

2.8 Decoding

Each sentence is decoded separately. The decoder considers all possible segmentations of the input sentence into phrases and tries to find a combination of phrase translations so as to maximize the overall probability of the hypothesis as defined in Section 2.6.

Since the problem of finding a globally optimal hypothesis is NP-complete, a variety of search algorithms were developed to find good sub-optimal solutions. We will describe the best-known algorithm *beam search*, which is used in the Moses decoder. We will also give a short overview of *cube pruning*.

2.8.1 Beam Search

In beam search, decoder generates the translation from left to right, keeping track of which parts of the input were already translated (using a *coverage vector*—a vector of Boolean values for each input word). Partial translations are stored in stacks based on how many input words they cover (i.e. the sum of the coverage vector). If the input sentence consists of k words, the decoder will create k stacks. The stacks have a limited size, the best-scoring hypotheses are on top. When a hypothesis does not fit in the stack, pruning occurs.

The hypotheses are also recombined during decoding. Recombination is done in two situations. The first case is very simple: whenever two identical hypotheses are generated (e.g. because of different segmentation), the one with worse score is discarded. However, hypotheses which are not identical can also be recombined. Suppose that the model uses a four-gram language model and that the last three words of two translations are identical. Then for the rest of the decoding, these hypotheses are identical (no scoring feature will look past the last 3 words). In this case, the lower-scoring hypothesis can be safely discarded as well (while keeping a back-pointer to retrieve it for n-best list generation).

The hypothesis score used in the stack is actually not the hypothesis probability from the log-linear model $P(h)$, but $P(h) + \mathrm{fc}(h)$, where "fc" denotes future cost. Future cost is an approximate score (log-probability) of the cheapest translation of the rest of the sentence. Intuitively, if a hypothesis translates a very easy part of the input sentence, it will have a high score, but also high future cost. Future cost prevents bad translations of easy parts from pushing good hypotheses (which got low score because they translated the difficult parts) away from the stacks.

2.8.2 Cube Pruning

Cube pruning is an alternative search algorithm designed originally for hierarchical decoding, but adapted for phrase-based machine translation in Huang and

Chiang (2007). Similarly to beam search, partial hypotheses are stored in stacks based on the number of covered source words.

At each iteration, a new stack is constructed, starting with the coverage of one source word and finishing when the stack for hypotheses covering the whole source sentence is built. Each stack is filled from a priority queue that contains partial hypotheses from previous stacks.

The pruning itself is inspired by parsing. Assume that we have two sets of hypotheses, each set covering a different part of source sentence, and that these parts are adjacent. Each set contains as many hypotheses as allowed (e.g. 1000) and our goal is to output the best 1000 combinations of the hypotheses. The score of combining two hypotheses is not simply a product because of features that cross phrase boundaries, most importantly the language model score. It is however still a good estimate to start with the best hypothesis from each set and combine them. We can then proceed greedily by trying the second-best hypothesis from both sets, finding out which combination (1-2, 2-1 or 2-2) has the best final score (including language model probability etc.) and expanding it. We repeat the process until we have greedily found the 1000 best combinations. This enables us to avoid constructing the whole space of 1 million hypotheses.

2.9 Evaluation of Machine Translation Output

Manual evaluation of translation outputs is very costly. Therefore, in most cases, researchers rely on automatic metrics of translation quality. Generally, a metric is a function which returns a score given a set of translated sentences and their reference translations[7].

2.9.1 BLEU

BLEU (Papineni et al., 2002), or Bilingual Evaluation Understudy, is the most popular metric today. It is used both for tuning translation model parameters, where it outperforms almost all existing metrics (Callison-Burch et al., 2011; Cer et al., 2010b), and for final evaluation of translation quality. Its suitability for the latter task has been disputed (Bojar et al., 2010 and others) and newer metrics show higher correlation with human judgement. Despite its limitations, it is still widely regarded as the de-facto standard method for machine translation evaluation.

BLEU is calculated as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.9}$$

N can be set freely, but typically the value 4 is used. The p_n stand for n-gram precisions, i.e. p_1 equals the proportion of words (unigrams) in the translation

[7]This definition does not always hold, there are metrics which do not use the reference translation, others incorporate also the source sentences.

output that were confirmed by the reference. Each word in the reference can be "mapped" to at most one word from the translation, making it impossible to artificially boost the score by e.g. outputting a frequent word many times. Precision values is weighted by w_n, but these weights are usually set to $1/N$.

To compensate for the focus on precision, the score is multiplied by brevity penalty (BP). Without it, it would be possible to get maximum BLEU score by outputting just one word that matched the reference (such as "the") and nothing else. Let c be the length of the candidate translation (i.e. machine translation output) and r the length of the reference. BP is defined by the formula:

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp(1 - r/c) & \text{if } c \leq r \end{cases} \tag{2.10}$$

There are two main disadvantages of BLEU for evaluating translation quality: BLEU favors translations with matching long word sequences and it disregards word structure.

In a free word order language such as Czech, keeping the exact ordering of words as the reference translation can be irrelevant to how well the translation conveys the meaning of the source sentence. This can result in translations beating an adequate translation that are worse overall but happen to keep a few words "correctly" ordered.

The second problem is even more pronounced when Czech is the target language. As Czech is morphologically rich, content words have many forms. The translation system often makes errors in choosing the correct form of the word. This error is evidently less serious than translating the word lexically incorrectly (often even sentences translated into lemmas are understandable), but BLEU disregards this difference.

Bojar et al. (2010) have shown that correlation between BLEU and human judgement is especially low when translations are very bad overall and set an empirical limit 20 below which BLEU score is not reliable. However, in this work, we will rely on BLEU (even below this limit) anyway, mainly because it is still considered a standard metric and as such gives the most informative picture of the achieved results. It is also well studied, so its problems are known.

2.10 Software

So far we have described the components of a machine translation system on a theoretical level. In this section, we provide an overview of the software tools used in this work.

2.10.1 Word Alignment

The toolkit we used for estimating the word alignment of the training data is Giza++ (Och and Ney, 2000). The authors incorporated a set of improvements, such as modeling fertility based on word length or smoothing. Even though

Giza++ is free, open-source software, several forks of it exist, most notably MGiza++ (Gao and Vogel, 2010) which adds support for multi-threading and iterative training, making it possible to align a new, small set of data given an existing alignment model.

2.10.2 Language Modeling

We used SRILM (Stolcke, 2002) for estimation of n-gram language models. Since 2010, Moses is distributed with KenLM (Heafield, 2011), which does not create language models, but allows for very fast and efficient model queries. This quality is essential for phrase-based decoding, as all generated partial hypotheses need to be scored by the language model for re-ranking. KenLM also stores language models in memory very efficiently, enabling the use of large models.

2.10.3 Decoding

Several free open-source implementations of phrase-based MT exist—the most active and widespread is Moses (Hoang et al., 2007), which we use in our experiments. cdec (Dyer et al., 2010) implements a variety of translation paradigms, including phrase-based as well as hierarchical translation models. Joshua (Li et al., 2009) is a widely used decoder for hierarchical machine translation based on synchronous context-free grammars. Phrasal (Cer et al., 2010a) is a recently developed decoder, notable for its support of discontinuous phrases, which enable it to handle reordering phenomena beyond the reach of hierarchical systems while maintaining its phrase-based nature.

Moses is distributed as a part of a toolkit for machine translation. It includes scripts and programs for phrase extraction, scoring, pre-processing of corpora, system training, tuning of translation systems and their evaluation. We use these tools extensively in our work, including the implementation of MERT and PRO and the BLEU metric.

Chapter 3

Factored Machine Translation

3.1 Introduction

Factored models are an extension of phrase-based translation. They were introduced in Koehn and Hoang (2007) with the aim to reduce several problems of the paradigm, centered around the inability to handle linguistic description past surface forms. In a factored model, the system no longer translates words. Instead, each word is represented by a *vector of factors* that can contain the surface form, but also lemma, word class, morphological characteristics or any other information relevant for translation.

Aside from a factored parallel corpus, a configuration of the model is needed for system training and subsequent translation. It is necessary to specify *translation* and *generation* steps which will produce the translation.

Translation steps map a defined subset of source factors to a defined subset of target factors. The translation proceeds similarly as in the phrase-based scenario, it operates on phrases.

Generation steps operate on the target side, their input is a subset of factors (already generated, e.g. by a previous translation step) and they output another subset of target factors. Generation operates on target words, so no word alignment is necessary. In fact, additional monolingual data can be used in their training.

The example in Figure 3.1 shows a scenario with two translation steps and one generation step. Source lemmas are translated to target lemmas, similarly for tags. The joint information is then used on the target side to generate final surface forms. Note that factored models used in practice are *synchronous*—the same segmentation into phrases is used for all translation steps.

Computing the mapping steps during decoding would be time-consuming. To make the translation more efficient, the decoder computes all possible phrase *expansions* ahead of time and stores them as translation options. This results in the reduction of factored decoding to the original algorithm for phrase-based translation.

Factored models, especially the more complex setups, can dramatically increase the computational cost—the combination of translation options of various

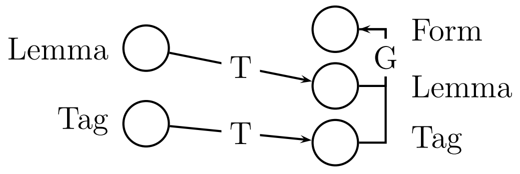


Figure 3.1: Factored translation. An example of translation and generation steps.

steps can cause a combinatorial explosion. Generating all of them is costly in terms of computational time and memory. During decoding, pruning will likely discard good hypotheses, as stacks will be filled with too many factor combinations.

Let us clarify some terms used in the following sections.

Hypothesis is the result of decoding, a candidate translation of a whole sentence.

Partial hypothesis is a translation of a part of a sentence. Translation is constructed left-to-right.

Translation option is a possible translation of a particular source phrase.

Partial translation option is a translation option that is not fully constructed. This term is specific for factored setups, where translation options are constructed by gradually filling in target factors according to the defined mapping steps.

Phrase expansion is the application of mapping steps on a source phrase. During phrase expansion, partial translation options are generated until all mapping steps have been applied, resulting in a set of (full) translation options that will be used during decoding.

Hypothesis expansion refers to extending a partial hypothesis by translating another part of the sentence. This is the key operation used in the search.

3.1.1 Translation Options in Factored Models

Before the actual decoding, Moses creates all possible phrase expansions and stores them as translation options. It means that the various mapping steps (i.e., translation or generation) are applied in the order specified in configuration. Consider the example shown in Figure 3.1. This particular translation system uses two translation tables (lemma→lemma, tag→tag) and one generation table (target lemma+tag→form). For each source phrase, Moses generates all possible translations of the lemmas. Then it combines each lemma with all possible translations of the tags that have the same length (resulting in a subset of cross-product of the lemma/tag options). Finally, each combination generates zero, one or more target forms.

Note that the order of translation steps does not affect the final number of translation options in this case—since the second step only considers the tags (it does not use the target lemma in any way), the outcome of the previous translation step has no effect on the options generated by this step. The number of generated translation options would be the same if we swapped the order of applying these two steps[1]. For the sake of completeness, let us note that moving the generation step before the translation steps would cause an error, since the required input factors would not yet be ready.

The effect of the step order for phrase expansion becomes more interesting if the steps share some of the output factors. In this case, only *consistent* translation options can be generated during expansion. An expansion is considered consistent if the output factors it generates match the already created factors. This restriction has two effects for phrase expansion. First, it limits the number of translation options generated from the existing options. Second, it discards those partial options for which no consistent expansion exists.

For example, suppose that we define two separate translation steps:

1. lemma→lemma
2. tag→lemma

Consider the phrase expansion in this order. First the possible translations of source lemma will be generated, resulting in a set of target lemmas. Then, consistent expansions will be considered. The lemma is already generated, so in fact, the second translation step will simply cause some of the target lemmas to be filtered out (presumably those that are bad translations given the tag, e.g. a morphologically ambiguous English word translated into Czech as noun instead of verb).

If we invert the order, the phrase expansion will proceed quite differently. First all possible translations of source tag into target lemma will be generated. The number of translation options will be enormous (we are asking the decoder e.g. for all Czech phrases that are translations of English words with tags "DT A N"). These options will surely be pruned, very likely discarding the correct translation. The consistency condition for the second step will then filter out the rest as nonsensical translations. It is evident that such factored system would not be very successful.

3.1.2 Decoding Paths

Apart from the rich possibilities of factored setups, Moses allows for multiple decoding paths to generate translations. Each path can be an independent, possibly factored model. The only requirement is that the paths accept the factors of the input and generate the factor(s) required in the output translation.

[1]Pruning may however discard different expansions if the steps are swapped—there are probably less options when translating lemma→lemma than tag→tag. Therefore if we swapped the order of the steps, first some valid tags may be pruned out.

There are two ways of integrating decoding paths:

1. Alternative paths.
2. Back-off models.

In the first case, both paths are equivalent and compete during decoding. Their translation options are implicitly combined to generate the most probable translation possible. If both models propose the same translation, both versions are kept along with their (almost certainly different) scores. Alternative decoding paths are commonly used to translate directly form→form and also decompose the translation into a series of factored steps that make the model more robust. On the other hand, form→form translation can be used as back-off from translation of more complex factor combinations, such as form|tag|afun → form.

The second scenario is useful when one translation table should be preferred. The other one is then consulted only in cases when a phrase is out-of-vocabulary. The user can specify the maximum allowed phrase length for the other table—if set to 1, it will only be used to fill in individual unknown words.

3.2 Factors

One of the aims of this work is to evaluate the benefits of incorporating rich linguistic annotation into statistical machine translation. We use Treex[2], a modular framework for natural language processing, to analyze our data. Treex performs morphological analysis, tagging, parsing and tectogrammatical annotation on both sides (English and Czech), enabling us to work with a wide range of linguistic information.

In this section, we will describe factors that we get directly from Treex as well as some new factors designed as a part of this work.

3.2.1 Extracted Factors

From the morphological layer, we extract the *lemma* and *morphological tag* of each word. Czech lemmas are disambiguated. English tags come from the Penn Treebank tagset (Santorini, 1990), Czech tags use the positional system of the Prague Dependency Treebank 2.0 (Hajič et al., 2006). This tagset is much richer than the English counterpart—about 2000 of the possible tags were seen in a corpus. When translating from English into Czech, morphological tag on the source side can provide valuable hints for the translation system, mainly for disambiguation of parts of speech.

On the analytical layer, dependency structure is the most important type of annotation, however it cannot be directly turned into a factor. We designed some custom factors that take the structure into account. Words on this layer are also annotated with their *analytical function*. A list of the functions can

[2]`http://ufal.mff.cuni.cz/treex/`

be found in annotation manuals for PDT[3] and PCEDT[4]. In fact, all subsequent factors discussed in this section are documented in these references. Examples of analytical functions include `Sb` for subject or `Pred` for predicate. When used as a factor, we can easily imagine that it can be beneficial for translation quality—for example, by distinguishing subject from object, we can help the decoder decide the correct case of the Czech translation.

The tectogrammatical layer describes the deep syntactic structure of sentences and discourse. It contains annotation of phenomena that border on the syntax and semantics, such as semantic roles, coreference or valency. From technical perspective, only content words are represented, while auxiliary words are mapped to content word nodes. Nodes that do not correspond to any surface word can be added on this layer, e.g. for omitted subjects in Czech or complement arguments.

We draw a number of factors directly from the annotation.

t-lemma Tectogrammatical lemma.

functor Describes syntactic-semantic relation of a node to its parent node. Its possible values include ACT (actor), PAT (patient) or ADDR (addressee). Functors could be a valuable source of information for the decoder, esp. for disambiguating syntactic roles—many phenomena which are represented in Czech by morphology are described by syntax in English and functors capture syntax on a deep level.

grammateme A set of factors that describe morphological properties of t-nodes. We extract each such category separately. Morphological features are a naturally useful resource for machine translation. We extracted the following grammetemes:

- **gender** Grammatical gender. In English, this factor is set only rarely, but in these cases, it may have the potential to disambiguate among Czech candidate surface forms.
- **number** Grammatical number. This morphological category is modeled explicitly in English (by adding “-s”). There might however be some benefit to the robustness of our statistical models if, for example, we decompose translation into mapping of lemmas and transferring the information about number separately.
- **sempos** Semantic part of speech. This factor is more fine-grained than ordinary part-of-speech tags. Its values include “n.denot” for denotative nouns or “adj.pron.def.demon” for definite demonstrative pronominal adjectives. This form of linguistic description may prove interesting for machine translation.
- **tense** This attribute specifies tense of verbs. This category is also modeled explicitly in both English and Czech. The argument we made for

[3]`http://ufal.mff.cuni.cz/pdt2.0/doc/pdt-guide/en/html/`
[4]`http://ufal.mff.cuni.cz/pcedt2.0/en/`

including the grammatical number holds in this case as well. For tense, there are also other possible ways of improving machine translation: the identification of infinitives (especially in infinitive clauses where it is possible to put words between the particle "to" and the verb) and disambiguation of uses of the gerund verb form in English ("ing").

verbmod This factor indicates the verb mood. For most verbs, this is "ind" (infinitive or indicative). For conditional mood, the value is "cond".

negation This is an indicator of negation for nouns/adjectives which begin with prefixes such as "un" or "ir". In some cases, negation in verbs is also marked.

formeme Contains a projection of some morpho-syntactic information from the morphological and analytical layers. It it mostly a technical solution aimed at simplifying search on the t-layer, but the specific aggregation of attributes that constitute formemes could be beneficial for translation.

3.3 Related Work

In this section, we set out work into the context of related research. We focus on enriching statistical methods with linguistic information, specifically using factored translation models.

Factored translation models were introduced by Koehn and Hoang (2007). The goal was to decompose the translation process into subsequent steps that gradually model different aspects of the translation until the surface forms are generated. Morphology was given as a motivating example and the original paper documents experiments with factors that correspond to morphological features. For English $\rightarrow$ German, large training data were used (750k parallel sentences) and the gains in BLEU were the slight, an improvement in modeling was however confirmed by manual evaluation.

Factored models for English $\rightarrow$ Czech translation were evaluated in Bojar (2007). Three factored configurations are proposed in this work:

T+C Surface form is translated to target form. A generation step then produces the morphological tag. Two language models (on forms and on tags) then rescore the hypotheses, promoting the grammatically coherent ones.

T+T+C This configuration extends the previous one by adding one final translation step that maps source tags to the target tags.

T+T+G The last setup is the most linguistically motivated. Lemma and tag are translated separately and one generation step is then used to produce the surface form.

All of the described models outpeform the baseline setup. However the last configuration fails to outperform the less complex setups. Additionally, this work includes experiments with various levels of granularity of the target-side morphology. The best result is achieved with an optimized tagset with roughly 1000 unique tags observed in the data.

Since their introduction, the use of factored models has been reported and evaluated in numerous research papers.

Cettolo et al. (2008) included shallow syntax into phrase-based machine translation using factored models. The additional factors captured syntactic chunks; more specifically, the source-side surface forms were translated into a combination of target form and microtag. Microtag contained also information about the chunk boundaries. Additional language models then checked both the probability of the tags as well as the chunking. Minor gains in BLEU scores are reported in the paper.

Another way of introducing syntax to phrase-based translation using factored models was described in Birch and Osborne (2007). The factored setup was quite simple, the surface form was mapped directly to a combination of form and a CCG supertag. These tags are related to Categorical Combinatorial Grammar and partially describe the part of the parse tree related to the given word. In this setup, the authors essentially mapped source surface to target surface form plus shallow syntax. For the evaluated language pairs (Dutch-English and German-English), the authors achieved improvements in the BLEU score.

Factored models have even been used for domain adaptation of existing models for machine translation (Niehues and Waibel, 2010). The central idea of this work is to use an additional factor that described where the word was taken from, its value is either "IN" or "OUT". The authors report gains in BLEU around 1 point absolute.

Chapter 4

Automatic Generation of Factored Setups

4.1 Estimating the Complexity of Factored Setups

An essential part of automatically creating and evaluating factored machine translation systems is the estimation of their computational cost. We want to know ahead of time whether a factored setup is feasible. Otherwise, we might spend a lot of time and computational resources on evaluating a configuration which cannot work in practice.

This is determined by a number of conditions, some of which are technical issues, such as memory requirements, the disk space needed or the computational time that it would take to train and evaluate a system.

But factored setups in particular suffer from combinatorial explosion of translation options during phrase expansion. Each translation step can potentially generate a full cross-product of translation options. These options have to be pruned (pruning occurs when translation options are created and during decoding); if their number is too high, the decoder is forced to discard even correct translations, resulting in search errors.

Furthermore, setups that have a high number of possible translation options also exhibit the technical problems—they run for a very long time and consume large amounts of memory and disk space. This combinatorial explosion is the primary criterion of feasibility of factored systems.

We therefore developed a tool that estimates the number of partial translation options generated by each step and use its approximations to avoid training of such unrealistic setups. The estimation is available for the whole setup, but also separately for each step, which can provide insights for analysis of factored models.

4.1.1 Estimation of the Count of Translation Options

There is no standard way how to obtain the number of partial translation options that a particular factored setup will generate. An exact estimate can be obtained by training the full translation system, translating some large and diverse test set and monitoring how many partial translation options have been generated for decoding. However, we need to approximate this number ahead of time and without much computation, in order to make automatic search for factored systems possible.

We are going to estimate this number by creating all translation/generation tables required by the configuration, but on a small sample of data. We will try to approximate the average and maximum number of partial translation options generated by each mapping step and overall, by all the steps combined. We will discuss approximating the average, the procedure for maximum is a simple modification.

One Translation Step

Assuming that we have generated the sample phrase tables, let us start with a simple example: a non-factored system with just one phrase table translating lemma→lemma. We might decide to simply calculate the average number of translation options per source phrase and declare the result our estimate, but this approach fails to predict the average number of translation options actually used during decoding.

The simple arithmetic average cannot be used because extracted phrases obey the power law in a sense. Evidently, phrases that occur only once have only one translation in the phrase table. These phrases actually make up most of the phrase table but in fact they are almost never used. We therefore use frequency-weighted average over all source phrases to estimate the number of translation options. Let t_i be the number of (distinct) translations of phrase i and let f_i be its frequency in the sample training data. Then the weighted average is:

$$avg = \frac{\sum_i f_i \cdot t_i}{\sum_i f_i} \tag{4.1}$$

Multiple Mapping Steps

Figure 4.1 shows an example of a scenario with two independent translation steps, lemma→ lemma and tag→tag. For each source phrase, Moses generates all translation options from the translation table that is defined first. Each of these partial options is then expanded using the second phrase table. Since the target factors are independent, all combinations of translations that have *the same length* are generated. It is apparent that approximating the number of full translation options as $avg_{t1} \cdot avg_{t2}$ would be an overestimate.

We will describe our approximation for mapping steps that follow the first one for a general case, where the steps are not even independent, i.e. they share some

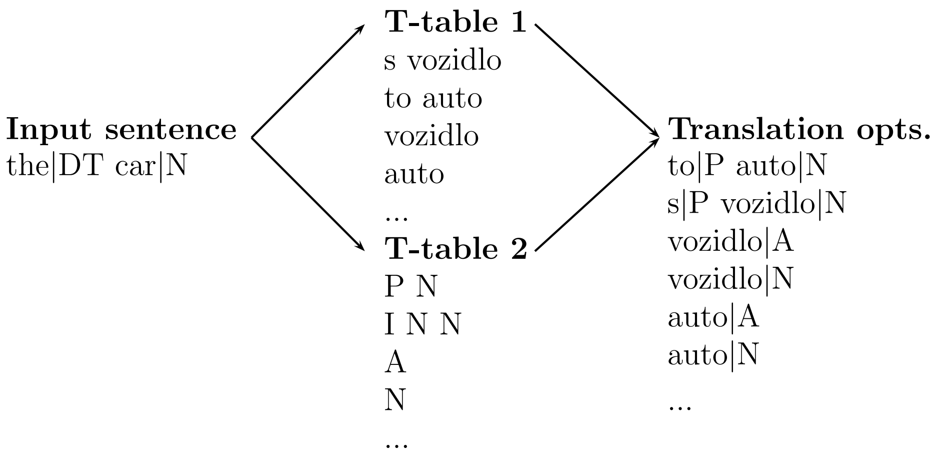


Figure 4.1: Phrase expansion in factored models. Options can be used multiple times, such as "DT N"→"N", or completely discarded if they are inconsistent, such as "DT N"→"I N N".

output factors. In this scenario, we need to take into account two constraints, the length and target factor consistency, in order to keep the estimate reasonably accurate.

Let us walk through the procedure of constructing one translation option. It is originally drawn from the first phrase table. This immutably defines its length and some target factors. In the following steps, the decoder therefore only chooses among translation options of this length and with the required output factor values. These steps add further constraints on the values of target factors.

To approximate this procedure, we factor each source phrase according to its length and the values of fixed target factors when counting translations. So each source phrase effectively becomes several source phrases, each with identical length and previously generated target factors. Figure 4.2 shows an example. We then count their translation options separately. When estimating frequency, we found that keeping the frequency of the original source phrase leads to better estimates.

To obtain an estimate of the number of full translation options, we simply multiply the estimates of all steps. Note that if the phrase table is sorted, the estimation of each step is linear with respect to phrase table size, so this algorithm is quite efficient.

Our heuristic does not completely capture the process of phrase expansion. We did not find a way how to estimate the effect of *implicit* pruning: for example, we might have a step that translates tag → tag and a following translation step form → form|tag. Some of the previously generated tags will be discarded (if the second step did not generate them) and some of the expansions as well (if their tag was not generated by the previous step). We did not find a way of estimating the amount of pruning involved without running a decoder.

```
the cat ||| kočku|N
the cat ||| kočkou|N
the cat ||| kočce|N

the cat ||| a|J kočka|N

the cat ||| tu|P kočku|N
```

Figure 4.2: Translation options for the phrase "the cat" divided according to length and values of target factors, assuming that target part of speech is generated by a previous mapping step. Instead of adding 5 to the average number of translation options, we add 3 source phrases with 3, 1 and 1 translations.

Generation Steps

So far we have discussed how to approximate the number of translation options for translation steps. Generation steps are slightly different as generation is done word-by-word. This implies that for a phrase of length k, there will be avg^k translation options. Again, we are forced to approximate in order to get an average number of translation options per phrase. We estimate the average k as the average length of target phrases in the first phrase table (since the first translation step sets the option length). The factor consistency constraint still holds and is taken into account similarly as with the translation steps.

Our prediction tends to heavily overestimate the explosion caused by generation steps. For example, if we have two previous translation steps, lemma $\rightarrow$ lemma and tag $\rightarrow$ tag, we can (quite precisely) estimate how many options the generation step lemma|tag $\rightarrow$ form will generate for each *surviving* partial option. The problem is that most combinations of lemma and tag will be discarded because they were not seen in the training data. This is the same problem as the implicit pruning mentioned above. Only in this case, its effect is more serious.

Full-Sized Data

The last issue is how the number of translations per source phrase changes when more training data are added. While the ratio of extracted phrases per training sentence pair remains constant if we add more data, the number of translations per source phrase slightly increases. For a phrase table trained on 5000 sentence pairs, the ratio in our experiment was roughly 1.64. For 200 thousand sentences, it increased to 2.17 and for half a million sentences, it reached 2.40. The training data were lowercased English and Czech forms from CzEng 1.0. We use random 5000-sentence samples from the training corpus to calculate our prediction. In our experiments the training corpus was roughly 200 thousand sentence pairs, so we set the ratio to $2.17/1.64 \doteq 1.3$.

Evaluation

We evaluated the estimation accuracy for several factored systems. We edited Moses to output the average number of translation options and compared the results obtained when translating a test set with our prediction. Table 4.1 shows the results.

Mapping Steps	Estimation	Moses Avg.
t:form→form	$1.3 \cdot 5.38 \doteq$ **7.0**	**12.4**
t:tag→tag +	$1.3 \cdot 11.28$	
+ t:form→form|tag	$1.3 \cdot 1.28 \doteq$ **24.4**	**84.6**
t:lemma→lemma +	$1.3 \cdot 5.23$	
+ t:tag→tag +	$1.3 \cdot 57.25$	**173**
+ g:lemma|tag→form	$1.3 \cdot 1.13 \doteq$ **655**	
t:lemma→lemma +	$1.3 \cdot 5.19$	
+ t:functor→functor +	$1.3 \cdot 52.48$	**5153**
+ g:lemma|functor→form	$1.3 \cdot 16.54 \doteq$ **9903**	

Table 4.1: Estimation of the number of translation options per phrase.

The translation form → form is simple and its estimation is quite close to the real value (if we used the test set, we could be almost completely accurate, however we only used the training data—other test sets can lead to different numbers of translation options). As we progress to more complicated setups, the results start to suffer from the deficiency of the heuristic (as discussed above). However we can see that still, we are roughly (in the orders of magnitude) able to predict the number of translation options. It is also important to note that while the precise values are wrong, the ordering of the setups remains the same. This allows us to use the heuristic to discover the difficult systems. For example, the last setup (with functors) ran many times longer than the identical configuration with tags (despite the fact that there are far more tags than functors). This difference is correctly reflected by the heuristic.

Overall, we were not able to predict the complexity with the precision we aspired to, however the basic goal of our work was met: we have a means to pinpoint difficult configurations without running the experiments. Moreover, the decomposition provided by the heuristic allows us to identify the difficult mapping steps.

4.2 Generation and Management of Experiments

In order to explore the space of possible factored configurations, we need to carry out a large number of experiments. We create configurations for these experiments automatically by declaring a set of steps (such as word alignment or language model training) and variables along with their possible values. Steps can be combined in a defined way and some variable values interact with each other, which is handled by placing constraints on them. Our tool Prospector then automatically searches for the best configuration by creating and running the experiments.

4.2.1 Eman

A pre-requisite for Prospector is a framework for experiment management. We decided to use eman, which is a tool developed exactly for this purpose. In this section, we will introduce some key concepts of eman which are relevant to this work.

Eman operates in a directory called "playground" where it creates the requested steps. Each step is defined by a seed, which can be any executable file that prepares the step code (stored as "eman.command"). Steps can have variables that define their behavior and they can depend on each other. The combination of seed and variable values defines the step. Figure 4.3 shows variables of a particular instance of seed `align`.

```
+- s.align.e357fb70.20120221-1115
|  | ALILABEL=en-lemma-cs-lemma
|  | ALISYMS=gdfa
|  | CORPUS=czeng-news
|  | GIZASTEP=s.mosesgiza.fcfbe812.20120221-1114
|  | SRCALIAUG=en+lemma
|  | TGTALIAUG=cs+lemma
```

Figure 4.3: Variables of a single eman step.

In this context, a machine translation experiment can be seen as a directed graph where steps with given variable values are nodes and their dependencies are arcs. The whole scenario can be compactly represented in a "traceback", which contains all the information necessary to reproduce the experiment. An example of a traceback is shown if Figure 4.4.

```
+- s.evaluator.7615f4de.20120510-1222
|  +- s.translate.35d0e771.20120510-1222
|  |  +- s.mert.d18c8ff5.20120510-1221
|  |  |  +- s.model.6c65e872.20120506-2333
|  |  |  |  +- s.lm.31e6f9e1.20120221-1114
|  |  |  |  +- s.tm.47378aea.20120506-2333
|  |  |  |  |  +- s.align.e357fb70.20120221-1115
```

Figure 4.4: Example of eman traceback.

Eman handles the execution of steps (either locally or as jobs on a cluster) and ensures that the dependent steps will be run in the correct order. If a step fails, it can be re-run (the user can correct the step code, eman.command, in the meantime). Essential features of eman are step re-using and cloning of steps or whole tracebacks. When the user asks for a step to be created, eman first determines whether such a step already exists. If so, it will automatically use it, avoiding the unnecessary repeated execution. Moreover, the user can ask for a whole experiment at once by supplying the full experiment traceback. In that case, eman decides which of the steps actually have to be run and only creates these steps. The existing ones are re-used.

Given an existing experiment, we might like to change some variables and observe the results. This is supported by eman via cloning of individual steps or whole scenarios.

Eman is purpose-agnostic, but it is used primarily for managing machine translation experiments. Therefore all the necessary seeds are already developed and well tested in practice. This includes seeds for corpora preparation, word alignment, model training, tuning, evaluation and many others. The "SMT playground", in which these seeds are defined, also contains a number of valuable tools for machine translation. One notable example is corpman, a tool for managing and generating corpora used by many of the SMT seeds.

4.2.2 Prospector

We developed Prospector for this work, but its interface is very flexible, so it will hopefully be used in further research in machine translation to simplify the search for optimal system configurations. Prospector builds upon the capabilities of eman and it is fully integrated into eman's concepts, which allowed for relatively simple and straightforward implementation. Prospector is written in Perl 5 for Linux. A user manual is provided in Chapter A.

Variables

The set of experiment configurations for Prospector is defined by an eman traceback, variables and their possible values and a set of rules that define constraints on them. The traceback contains marked slots for specific variables which Prospector fills in to create a specification of a new experiment. Figure 4.5 shows an example traceback of step `align` with Prospector variables specifying source and target factors on which the word alignment should be computed.

Rules

The user defines possible values for both variables (such as lemma, surface form or pseudo-stems of various lengths) and optionally rules. Rules restrict possible combinations of variable values and have the following form: if the value of variable X matches a regular expression R, then the allowed values for variable Y are Y1, Y2, Y3. In order to restrict more than one variable, the user can simply add

```
+- s.align.e357fb70.20120221-1115
|  | ALILABEL=en-#SRCFACTOR#-cs-#TGTFACTOR#
|  | ALISYMS=gdfa
|  | CORPUS=czeng-news
|  | GIZASTEP=s.mosesgiza.fcfbe812.20120221-1114
|  | SRCALIAUG=en+#SRCFACTOR#
|  | TGTALIAUG=cs+#TGTFACTOR#
```

Figure 4.5: Excerpt from a traceback with variable slots for Prospector (surrounded by '#').

another rule with the same condition. In our experiments, rules are useful e.g. when specifying translation steps—their allowed values depend on the number of source factors.

Search Algorithms

In more complex experiment scenarios, there may be too many combinations of variables to evaluate. Prospector therefore offers several search algorithms that guide the generation of new experiments. In order to use these algorithms, the user has to specify a scoring function that will, given a finished experiment, output its score, or "fitness". Technically, this function is an executable file in the configuration directory that takes the identification of the final experiment step as argument and outputs a single number. In our case, this is a simple bash script that reads the BLEU score from the evaluation step.

We will now describe the implemented algorithms.

exhaustive Evaluates all possible combinations of variable values allowed by the rules.

line exploration Starts with all variables set to the first value (assumed to be a default value). Then for each variable separately, the algorithm evaluates all its possible values (keeping the other variables set to default). After performing the search for all variables, it combines the best values of each variable (as evaluated by the user-defined scoring function) into a final experiment scenario.

random At each step, sets all variables to a random value and evaluates the combination (again, if the rules allow it). Stops after generating a defined number of configurations (50 by default).

genetic Progresses in generations. Its parameters are the number of configurations in one generation, the number of best configurations that should be combined to create the next generation, and the probability of mutation. This is a standard genetic algorithm: the first population is generated randomly. All configurations are evaluated by the scoring (or fitness) function.

N best configurations are then selected. Configurations in the next generation are obtained by performing a cross-over of two randomly selected parents (from the N best): each variable has a probability 0.5 to be inherited from either parent. Mutation can occur on any variable, resulting in setting it to a random value. Similarly to previous algorithms, variable values must be allowed by the rules, random generation is therefore repeated as many times as necessary to produce the defined population size.

Prospector repeatedly checks the status of created experiments. New experiments are created only when the number of configurations already running is below a defined limit, in order not to overwhelm the cluster. Failed experiments are not automatically restarted—it is difficult to assess whether the failure was due to some random technical problem or whether the experiment is defined badly. This decision is left to the user.

Thanks to eman's efficient experiment cloning, the overhead of creating an already existing experiment is minimal—all the existing steps are simply re-used and eman outputs the finished final step immediately. Prospector can therefore be re-run (for example, after a crash) without concern.

Prediction of Setup Complexity

The user can optionally define one more function: a predictor of experiment complexity. Along with this function, the user supplies a threshold value beyond which the experiment will not be created. Naturally, the predictor used for our experiments is the estimation of average number of translation options, as described is Section 4.1.

This function plays a key role in enabling the fully automatic search for factored setups: many configurations are intuitively nonsensical (such as translating person|tense→form), but without the prediction function, Prospector would have no information that could prevent such a scenario from being generated and evaluated (which would presumably result in Moses running out of memory or in BLEU score around 0). A scenario such as this one has an intractable number of possible translation options that surely exceeds any sensible threshold value, so Prospector will discard it based on this prediction.

4.3 Search in the Space of Factored Configurations

Even though the space is discrete, there is an enormous number of possible configurations. In general, a machine translation system can contain any number of mapping steps (translation or generation). Each translation step can map any subset of source factors (2^s possibilities) to any subset of target factors (2^t possibilities). Similarly for generation steps that map target factors to other target factors. In principle, we cannot even limit the number of subsequent mapping steps, aside from disallowing an identical mapping step to be done more than once (even though in practice, we can hardly go beyond just a few subsequent steps).

4.3.1 Search Constraints

We can safely set a few constraints that can limit the explosion of possible configurations, but even so, the space is prohibitively large. Namely, we can assume that:

1. There exists a mapping step that creates the target form (and possibly some other factors).
2. There exists a translation step which contains either lemma, tlemma or form on both sides (steps such as "form|tag→tlemma|tag", but not e.g. "form→ tag|gender"). In systems where such a step is absent, the lexical information cannot be transfered and very poor translation quality can be expected.
3. Generation steps only use existing factors (i.e. factors that were previously created by either translation or generation steps).
4. Let $f_s(A)$ and $f_t(A)$ denote the set of source and target factors for step A, respectively. Then for each two steps A, B in one decoding path, the following condition holds:

   $$f_s(A) \nsubseteq f_s(B) \vee f_t(A) \nsubseteq f_t(B)$$

   In other words, A is not subsumed by B.
5. For each step, at least one of the factors produced by it is used in some subsequent step. The only exception is the target form if it is generated in the last step.

Note that the step which generates the target form is not necessarily the last one. The motivation for such setups can be twofold. First, it is possible to use language models on other factors than form. It is then conceivable that we would generate the target form in some intermediate step and then create more factors based on it (such as tag or various grammatemes). These could then be used to rescore the hypotheses. Another possible reason for generating the target form sooner involves pruning: for example, we might have a generation step that produces form and tag jointly and then a translation step mapping the source tag to the target tag. This step would filter out some unlikely tags before decoding (during phrase expansion).

The second assumption is empirical. If we evaluated all possible combinations and did not take this assumption into account, the prediction of setup complexity would discard setups that do not satisfy it anyway—the number of translation options would explode (some of the uninformative factors has to generate the target form, according to constraint 1).

Without the third assumption, we could generate erroneous setups which would crash immediately.

The fourth constraint prevents generating setups where one step is contained in another step. For example, consider a translation step form|tag → form|tag. If such a step exists, then steps such as form → form|tag have no effect (regardless of their position in the step sequence).

The last assumption is quite straightforward. Essentially, we require the mapping steps to have an effect on the translation—if all factors produced by a step are never used, we do not need to have the step in our configuration. Note that we do not require all factors to be used. For example, a step may generate (jointly) target tag and analytical function. Only the tag will then be used to generate the surface form. Then the role that analytical function plays in this step is to filter out inconsistent translation options—in a sense, by being "tied" to the target tag, it is not actually unused.

4.3.2 Enumeration of Possible Configurations

In order to explore the space of factored setups, we need a method for enumerating all possible configurations in some well-defined order. We (partially) order the factored configurations by dividing them according to the number of mapping steps. If we searched the space of configurations automatically, we would start with 1 mapping step. After exploring all possible options, we would move on to setups with 2 mapping steps (TT or TG), then 3 mapping steps (TTT, TTG, TGT or TGG) and so on.

For each number of mapping steps, these options can be obtained by generating all possible combinations and checking them against the search assumptions described above. If we aspired to a fully automatic search, we would then filter the remaining options based on the complexity prediction and run full experiments with the promising setups.

In this section we give an analysis of the number of possible (promising) setups. This analysis leads us to conclude that fully enumerating the space is not feasible.

One Mapping Step

Let us first discuss the number of possible configurations that have just one step. This is surely a translation step (implied by assumptions 1, 2 and 3). The step generates the target form[1]. On the source side, it uses at least one of these factors: form, lemma, tlemma. It can also use any number of additional factors. Because a set has exponentially many subsets, we can estimate the number of possible one-step setups as $O(2^s)$, where s represents the number of source factors. It is

[1]If we used multiple language models, we might even generate some other factors.

immediately apparent that we cannot hope to explore a representative subset of this space.

A possible solution is to evaluate the impact of each factor separately (in combination with source form) and then proceed to combine only the promising factors. However we show in Section 5.3 that the differences between factors are very small and the space that tuning needs to optimize in is complex. True improvements of translation quality are blurred by the random variance in tuning and overlapping confidence intervals. This prevents us from exploring the configuration fully automatically. We carry out some experiments in this direction while manually inspecting the intermediate results. Our findings are discussed in Section 5.5.

Multiple Mapping Steps

The situation becomes more complex if we move on to more than one mapping step. We cannot hope to explore longer sequences of steps if we allow any combination of factors. It is necessary to constrain the possible configurations even further. We tried to limit the maximum number of factors involved in each mapping step.

Let us analyze the number of possible setups if we set the limit to 2 and use two translation steps[2]. In the previous section, there was only one step, so naturally, it used all the available source factors (in the particular subset) and mapped them to all target factors (again, in the given subset). This case is not as straightforward, even though we only have two factors on each side. Let us assume, for simplicity, that the first factor (denoted by 0) on both sides is the surface form. The first translation step can use the following factors on the source side:

- Only factor 0 (form).
- Only factor 1 (an additional factor).
- Factors 0 and 1.

On the target side, the options are the same. The next translation step is constrained by the choice of factors in the first step, but still, there are multiple options in some cases. We manually enumerated the possible scenarios in this very small and restricted setting. After evaluating all constraints that were laid down in Section 4.3.1, we were left with a number of possible scenarios.

Regarding the values for factors, as stated previously, the factor 0 is always the surface form on both sides. Choice of the other factors is free (source factor 1 and target factor 1 can be different).

Table 4.2 shows the viable configurations. The first two columns show factors used by each translation step. For each combination, we provide an example of a

[2]This is not the simplest scenario possible: if the second step was a generation step, we would have considerably less options. However, the search algorithm has to evaluate both scenarios.

potentially good translation system, mainly to show that these combinations are not merely theoretical possibilities but distinct setups that warrant exploration.

In the last column, we attempt to manually estimate the number of combinations of factor values that should be evaluated for each setting. In our experiments, we limited ourselves to 12 additional factors, two of which are lexically informative (lemma, tlemma). Our estimation is based on this setting.

Steps		Example Setup		Estimated
First	Second	First	Second	Combinations
0→0	1→0	form→form	tag→form	12
0→1	1→0,1	form→POS	lemma→form|POS	48
1→0	0→0	lemma→form	form→form	2
1→0	0→0,1	lemma→form	form→form|tag	24
1→1	0→0,1	tag→tag	form→form|tag	144
1→0,1	0→0	lemma→form|POS	form→form	24
1→0,1	0→1	lemma→form|POS	form→POS	24
0→0,1	1→0	form→form|tag	lemma→form	144
0→0,1	1→1	form→form|tag	tag→tag	144
0,1→0	0→0,1	form|tag→form	form→form|tag	144
0,1→0	1→0,1	form|lemma→form	lemma→form|tag	144
0,1→1	0→0,1	form|tag→lemma	form→form|lemma	144
0,1→1	1→0,1	form|lemma→lemma	lemma→form|lemma	144

Table 4.2: Enumeration of 2-step mapping configurations.

We found 13 possible factored scenarios for two mapping steps and estimate that 1142 systems would have to be evaluated if our goal was to explore the space exhaustively.

In the first row of Table 4.2, there is only one free factor. We can set it to any value, so the number of possible systems is 12. The second row is slightly more complicated. Either the first or the second mapping step can transfer the lexical information. If it is the first step, then the second factor on the source side must be lexically informative (i.e. lemma or tlemma). Otherwise, it would be producing a lexical factor from a non-lexical one. The second factor on the target side is then free (12 options, 24 in total). For the second step, the situation is analogous, resulting in total of 48 options.

In steps where both factors are free, we (theoretically) need to evaluate $12^2 = 144$ system configurations.

There results show that exhaustive search is unrealistic even in this extremely restricted setting. Furthermore, adding other factors and/or mapping steps would cause this number to grow rapidly. At the same time, guided search is impractical due to the lack of reliable estimation of benefits of particular factors: the differences between translation systems are too small and hard to measure.

4.3.3 Decoding Paths

Another issue that we must address when enumerating possible factored setups are decoding paths. Very frequently, complex factored setups are backed off by a simple form→form translation step in an alternative decoding path (or vice-versa, if target forms are sparse, a factored setup might be used as back-off). When we generate more complex factored scenarios, some alternative decoding path might be appropriate.

As discussed in Section 3.1.2, the alternative path can be either a fully competing search (alternative path) or just a back-off for unknown words or word sequences (back-off path). In the latter case, the user can define the maximum phrase length that can be queried in the back-off table, giving yet another free parameter to explore.

One Mapping Step

In this scenario, we are able to enumerate all appropriate back-off steps for any number of source factors (again, we assume that there is only one target factor, the surface form).

Consider the following factored setup:

$$\text{form}|\text{formeme}|\text{tag}|\text{afun}|\text{sempos} \rightarrow \text{form}$$

The source side likely has a high out-of-vocabulary (OOV) rate—many of the factor combinations in the test set will not have been observed during training. The simplest (and possibly adequate) solution is to have a single alternative path: form → form. The full set of possible back-offs is shown in Table 4.3 (we list the source factors, all back-off paths translate into target form).

Given k source factors, we can express the number of possible back-offs simply as follows:

$$\text{backoff}(k) = \sum_{i=1}^{k-1} \binom{k-1}{i} \tag{4.2}$$

In our case: $\text{backoff}(5) = 4 + 6 + 4 = 14$.

Theoretically, we should include all of them—naturally, this would complicate both tuning and decoding beyond feasibility. In our experiments, we will provide evaluation of several back-off schemes, but we will not explore them automatically. Also, the question still remains whether we should use the alternative path or the back-off setting (and in that case, how long phrases we should allow). A brief evaluation of the back-off setting showed a decrease in BLEU. We therefore use the alternative path in our experiments; fixing the type of decoding paths reduces the number of setups that must be evaluated and enables us to go further in our experiments.

4 factors:	form|formeme|tag|afun
	form|formeme|tag|sempos
	form|formeme|afun|sempos
	form|tag|afun|sempos
3 factors:	form|formeme|sempos
	form|formeme|afun
	form|formeme|tag
	form|tag|sempos
	form|tag|afun
	form|afun|sempos
2 factors:	form|formeme
	form|tag
	form|afun
	form|sempos
1 factor:	form

Table 4.3: Back-off decoding paths for one mapping step.

Multiple Mapping Steps

If the translation is composed of multiple steps, it is not clear what the back-off should be. Removing any step or factor from the pipeline might break the setup or completely change its behavior. Given that we limit our experiments to steps that use at most two factors, we decided to use a single back-off strategy for all cases: an alternative decoding path with one translation step that maps source forms to target forms (and any other required target factors, when additional language models are used).

Note that if we translated in the opposite direction (from Czech into English), such a strategy would not be considered a back-off. The situation would be reversed: because Czech has a rich morphology, we might like to translate surface forms directly and in case of out-of-vocabulary words, we would back off to a factored setup (e.g. lemma|number → form).

Chapter 5

Experiments

5.1 Data

We used various data sets in this work. The corpus used for our initial experiments (sections 5.3, 5.2) was CzEng 0.9 (Bojar and Žabokrtský, 2009).

The main source of data for the remaining experiments is CzEng in its latest release 1.0 (Bojar et al., 2012a). It is a richly annotated Czech-English parallel corpus with over 15 million parallel sentences from 7 different domains: fiction, legislation of the European Union, movie subtitles, parallel web pages, technical documentation, news and manual translations from Project Navajo (originally machine-translated Wikipedia articles).

For most of our work, we do not use the whole CzEng, for practical reasons. Training a system on such amounts of data takes several days, automatic search for factored configurations would then be impractical. More specifically, we use the CzEng news domain as our source of both parallel data for translation model training and target-side monolingual data for language modeling.

Our development data (for system tuning) are the test set for WMT11 translation task (Callison-Burch et al., 2011). For final evaluation of each system, we use WMT test set for 2012 in its full size. The evaluation data for WMT are news articles, hence the choice of training data.

Table 5.1 shows basic statistics of the used data.

Data Set	Data Source	Sentences	En Words	Cs Words
Training	CzEng 1.0 news	197053	4641026	4193078
Development	WMT11 test set	3003	74822	65602
Test	WMT12 test set	3003	72955	65306

Table 5.1: Statistics of the data used in experiments.

5.2 Additional Factors

In our initial experiments, we analyzed a wide range of factors acquired by traversing dependency trees on both the tectogrammatical and analytical layer. We focused on direct neighbors, i.e. parent or child nodes (the first child in the surface word order), and extracted factors from their annotation using Richextr (Tamchyna and Bojar, 2010). Note that when we extract factors from the t-layer, we have two ways (not necessarily equivalent) of arriving at the child/parent node. We can either first move to the t-layer and look for the neighboring node there, or first go to the child/parent node and then move to t-layer. However, the differences in BLEU between these two options were negligible (as we expected), so we only report the results of the first path for consistency.

Factor	BLEU
child(0)→tlemma	25.10
functor	25.09
—	24.99
child(0)→lemma	24.92
formeme	24.92
sempos	24.92
parent→lemma	24.90
parent→tag	24.89
parent→functor	24.82
tag	24.81
child(0)→functor	24.73
parent→sempos	24.69
lemma	24.67
parent→nonterm	24.65
child(0)→nonterm	24.65
parent→tlemma	24.64
child(0)→formeme	24.64
child(0)→tag	24.60
tlemma	24.44

Table 5.2: Comparison of systems with factors obtained by traversing dependency trees.

While many of these factors seem appealing (such as part-of-speech tag of the parent node, or its analytical function), we did not manage to get significant improvement in BLEU score using any of them. We document the obtained results here, but in our main experiments, we did not use these factors because they would dramatically increase the number of possible factored configurations.

We used CzEng 0.9 data for training (200k sentence pairs), development and evaluation (1000 sentence pairs for each set). We always translated a combination of form and a second factor directly into target form with an alternative decoding path form→form. Table 5.2 summarizes the achieved BLEU scores (average from 3 runs of MERT). The setups are sorted by the BLEU score and the baseline system is denoted by '—'.

5.3 Evaluation of Factored Configurations

The simplest way of evaluating two translation systems is to translate a test set using both of them and compare the achieved BLEU scores. This straightforward procedure however disregards the fact that model tuning is randomized. For simpler translation systems, this is not necessarily an issue as the number of parameters is small. In factored systems, each mapping step has its own set of parameters (usually 5 for translation and 2 for generation steps), so the randomness becomes an important factor.

MERT uses random starting points to avoid reaching local optima (line search is globally optimal, but its result depends not only on the weight currently tuned, but also on the values of other weights). When the number of weights is small, this strategy gives good results that remain stable if we run MERT multiple times. However with factored models, the variance among MERT runs can be considerable.

PRO also includes some degree of randomness; at each iteration, n-best lists are sampled for pairs of translations with large differences in sentence-level BLEU score. This sampling is random (albeit guided by this measure of "fitness") and can result in variance of final BLEU score over the runs.

During this work, we ran several sets of experiments to determine the impact of randomness in tuning and the performance of tuning algorithms for our task. Our aim is to find an evaluation procedure that we could rely on for the automatic search in factored configurations.

5.3.1 Stability of MERT

As a part of initial research for this work, we carried out a number of experiments, exhaustively evaluating all two-factored setups. Some of the results are shown in Section 5.2, the used data are also described in this section. We used two alternative decoding paths, one that translated form|factor → form and another that only mapped form → form (as back-off). Each of these paths represents five weights that need to be optimized. Our initial goal was to evaluate the usability of each factor but the variations in results also lead us to question our evaluation methodology.

Table 5.3 shows a selected subset of the evaluated factors. Initially, we ran MERT once for each factor (see the BLEU column), keeping the default size of n-best lists output by the decoder. Surprisingly, all factors underperform the baseline (denoted by '—' in the Factor column), even though it was included in the factored setups as the alternative path. While it is possible that search errors contributed to these results, the inability of MERT to handle the complex space

Factor	BLEU	BLEU (300-best)	Max	Mean	StDev
child(0)→tlemma	24.77	24.75, 25.12, 25.43	**25.43**	**25.10**	0.28
functor	25.17	24.99, 25.03, 25.26	25.26	**25.09**	0.12
—	**25.22**	24.66, 25.15, 25.16	25.16	24.99	0.23
afun	25.06	24.62, 25.14, 25.21	25.21	24.99	0.26
formeme	24.54	24.58, 25.08, 25.09	25.09	24.92	0.24
sempos	25.11	24.75, 25.00, 25.01	25.01	24.92	0.12
parent→lemma	24.76	24.71, 24.97, 25.01	25.01	24.90	0.13
parent→tag	24.43	24.63, 24.83, 25.21	25.21	24.89	0.24
tag	24.73	24.61, 24.74, 25.07	25.07	24.81	0.19
lemma	24.66	24.34, 24.80, 24.88	24.88	24.67	0.24

Table 5.3: Statistics of BLEU scores achieved in multiple MERT runs.

of model weights is the key problem.

To evaluate the contribution of each factor more reliably, we increased the number of sentences in n-best lists to 300 (the default is 100) and ran MERT for each factor three times. The third column in the table shows the results of the three individual runs.

We can see that differences in BLEU scores in MERT runs are often as high as 0.5 absolute point, which is roughly the same as the improvement we expect from incorporating a useful factor in the system. Distinguishing between mere luck and true improvement is therefore difficult.

In fact, if we disregard statistical significance and look simply at the BLEU scores, we might draw very different conclusions depending on which MERT run we consider. We can even entirely invert the ordering of factors:

- tag (25.07) > functor (25.03) > sempos (25.01) > baseline (24.66)
- baseline (25.16) > sempos (25.01) > functor (24.99) > tag (24.61)

Moreover, if we use just one MERT run and do a statistical significance test, specifically the bootstrap resampling as introduced in Koehn (2004), the confidence intervals are so wide that we cannot consider any two systems to be significantly different.

There are several possible solutions to this problem. We can choose a different, more stable algorithm, and rely on the score of just one tuning run. The bootstrap resampling would still not distinguish between the systems, but we would have a better chance of finding out which factors are beneficial. To this end, we evaluated the PRO algorithm in Section 5.3.3.

On the other hand, if we decide to keep using MERT, we have to run it multiple times and then calculate the mean BLEU score and standard deviation (as we did in Table 5.3). Recently, pair-wise significance tests that sample from multiple runs of the optimizer have been suggested (Clark et al., 2011). The main purpose of this method is to decide in cases where the difference between two systems is small, whether it is statistically significant or not. This information is potentially interesting for our work, but we decided to use only simple statistics (mean and standard deviation), which provide sufficient information and do not require a pair-wise setting.

5.3.2 Space of Model Weights

The reason for MERT instability is not entirely clear. Our initial hypothesis was that MERT simply does not scale to a larger number of weights. In order to evaluate it, we created a synthetic experiment. The configuration was identical to the experiments described above, except that the translation table with an additional factor contained only one phrase. This phrase contained one word on each side, both of which were out of vocabulary. This way, MERT had to tune the same number of weights, but the space was greatly simplified: five weights were completely irrelevant (any phrase receives zero score from the synthetic translation table; changing the weight will not make any difference). Note that this setup is in fact identical to a baseline experiment—the only phrase table that is used translates source form to target form.

If we obtained a worse BLEU score, we could say that MERT truly scales badly to a larger number of weights. However, we obtained BLEU almost *identical* to the baseline experiment. Our conclusion therefore is that the complexity of the space is the main problem, not the number of weights per se.

5.3.3 Stability of PRO

We ran experiments with the same data and factors, but used PRO as the algorithm for tuning model weights. Reportedly, PRO is very stable, scales to a large number of features and achieves results similar or superior to MERT.

Our experiments confirmed the stability of the algorithm. The deviation in most cases was an order of magnitude smaller than with MERT (except for the *tag* factor, but in light of the other results, we can regard it as an outlier). However, notice that the order of factors in Table 5.3 and Table 5.4 is very different (both tables are sorted according to average BLEU score). Also, even though MERT is much less stable, it often finds a better set of weights than PRO.

To conclude, the stability of PRO can be a very useful quality for building machine translation systems. However, for the purposes of comparing systems which perform quite similarly, we cannot rely on the score achieved by PRO to distinguish between them.

We therefore decided to evaluate all of our experiments by running MERT several (3) times and calculating the simple statistics as shown in the previous sections.

Factor	Mean BLEU	StDev
tag	24.90	0.15
sempos	24.90	0.02
—	24.84	0.04
child(0)→tlemma	24.82	0.04
lemma	24.81	0.04
parent→lemma	24.80	0.03
formeme	24.79	0.02
parent→tag	24.71	0.02
nonterm	24.67	0.01
functor	24.56	0.02

Table 5.4: Statistics of BLEU scores achieved by PRO.

5.4 Parameters of Decoding

One direct benefit of Prospector is the possibility to exhaustively evaluate combinations of parameters passed to Moses. These are mostly of technical nature and modify thresholds for pruning, stack sizes etc. Previously, we would have written a script that would run all the required experiments. If we later decided to evaluate a different set of parameters, we would have to change the script code, debug it and run it again. With Prospector, carrying out such sets of experiments is very easy, which encourages us to perform evaluation of various settings that we would otherwise ignore.

The motivation for this set of experiments is to determine the impact of search parameters on translation quality. For factored setups, the explored space of hypotheses is arguably more complex than in baseline phrase-based setting. The decoding process might therefore be more prone to search errors, impairing the potential benefits of evaluated factors.

For these experiments, we use a typical, moderately complex factored scenario. We translate lemma to lemma and tag to tag (note that the English tagset is much smaller than the Czech positional tagset). We then generate surface form from the combination of target lemma and tag. As an alternative decoding path, we translate directly form to form. We use a language model on forms and a higher-order model on target morphological tags.

Data sets used for tuning and evaluation of these experiments are slightly different from those stated in Section 5.1: we used WMT10 test set for tuning and WMT11 test set for evaluation.

5.4.1 Beam Search

Beam search in Moses is driven by 2 main parameters:

stack Defines how many hypotheses can be stored on each stack. Pruning that occurs when this limit is exceeded is called *histogram pruning*. Setting the parameter to higher values increases memory usage and decoding time, but could result in fewer search errors. According to Moses documentation, the increase in decoding time is linear with respect to maximum stack size. The default value in Moses is currently 200.

beam-threshold Defines by how much a hypothesis can be worse than the best hypothesis in the stack. Pruning hypotheses with too different score is called *threshold pruning*. Again, a higher threshold means greater memory usage and decoding time, but possibly a reduction of search errors. Its default value is 0.000001.

Based on results reported in literature and default values in Moses, we chose a set of values for each variable and evaluated all possible combinations using Prospector. The achieved BLEU scores are summarized in Table 5.5. Some of our experiments ran for too long or crashed; their cells are marked by '—'. The ± sign denotes empirical 95% confidence intervals as estimated by bootstrap resampling (Koehn, 2004).

	0.00001	0.0001	0.001	0.01	0.03
200	13.38±0.48	13.59±0.48	13.35±0.47	13.40±0.48	13.36±0.47
500	13.53±0.47	13.47±0.47	13.53±0.49	13.44±0.47	13.22±0.48
750	13.28±0.46	13.38±0.46	13.29±0.48	12.92±0.43	13.40±0.47
1000	13.42±0.48	—	13.49±0.47	13.40±0.47	13.24±0.47
1500	13.65±0.48	—	—	13.43±0.49	—

Table 5.5: BLEU scores for various values of *stack* (rows) and *beam-threshold* (columns) parameters.

There is no apparent relation between the values of beam search variables and BLEU scores. The variations, albeit large, seem random and point again to MERT instability. We see no trend of increasing BLEU with larger values of either variable. This allows us to assume that search errors are not a major problem, at least for moderately complex factored systems. The situation might change if we move to larger datasets—ideally, a similar set of experiments should be run in that case.

If we compare the time it took the decoder to translate our test set, there are some evident trends (see Table 5.6). Because the decoding time is proportional to stack size, we can even use the results to infer which parameter causes more pruning. Variations in the times are to be expected; we run the jobs on a cluster with machines of varying speed and current computational load.

Evidently, setting the stack limit to higher values has a dramatic impact on translation time. We do not observe such a tendency in the values for beam threshold. We believe that the stack limit (at any setting) causes more pruning and beam threshold has little effect on the number of allowed hypotheses. This suggests that the scores assigned to the hypotheses are similar and the selection

	0.00001	0.0001	0.001	0.01	0.03
200	7.5	3.6	12.3	5.8	3.7
500	8.3	16.1	10.7	11.4	23.8
750	13.6	16.5	52.2	42.5	15.0
1000	35.8	—	18.4	28.2	20.7
1500	65.3	—	—	37.6	—

Table 5.6: Duration of translation in thousands of seconds for various values of *stack* (rows) and *beam-threshold* (columns) parameters.

of the best hypothesis is therefore difficult. A more rigorous examination of the decoding process would however be required to confirm this conjecture.

When the stack limit is set very high, the translation time becomes almost unacceptable. This has great effect on the duration of MERT tuning; in combination with a higher number of iterations, some of the tuning steps took almost a week of computational time to finish—much longer than our state-of-the-art submissions for WMT that used all available data sets.

5.4.2 Cube Pruning

The cube pruning algorithm is quite different from beam search, and so are its parameters. Limit for stack size still applies, however it has no effect on translation time. It is recommended to set it to a high value in order to avoid unnecessary pruning. When we evaluated the other parameters, we therefore set it to 5000 for all experiments.

We evaluated the following parameters:

cube-pruning-pop-limit Defines how many hypotheses can be popped onto each stack. During decoding, best hypotheses are drawn from a priority queue and put on the current stack, this options limits their number. The default is 1000.

cube-pruning-diversity By default, hypotheses are divided into stacks based on *how many* words from the input they cover, not *which* words they cover. Setting this options to k ensures that at least k hypotheses for each coverage pattern are popped onto the stack.

Table 5.7 summarizes the achieved BLEU scores. Similarly to beam search, we can say that default values are sufficient for our experiments, i.e. that setting the options to less restrictive values does not increase the BLEU score.

Diversity does affect translation time. By setting it to higher values, we force the decoder to generate hypotheses that it would not consider otherwise, artificially extending the search space. On the other hand, higher pop limit does not seem to affect the translation time at all.

	1000	2000	3000	4000	5000
0	13.33±0.46	13.57±0.47	13.41±0.47	13.47±0.47	13.30±0.47
5	—	13.35±0.49	13.36±0.48	13.63±0.48	13.42±0.47
10	13.33±0.47	13.37±0.46	13.54±0.48	13.49±0.47	13.42±0.46
50	—	13.53±0.45	13.48±0.46	13.26±0.44	13.27±0.47
100	13.35±0.48	13.12±0.47	11.83±0.46	12.54±0.46	13.41±0.47

Table 5.7: BLEU scores for various values of *pop-limit* (columns) and *diversity* (rows) parameters.

	1000	2000	3000	4000	5000
0	0.9	1.4	1.3	1.1	1.0
5	—	1.4	1.3	1.5	1.0
10	1.1	1.7	1.6	2.3	1.8
50	—	1.9	2.6	2.5	3.1
100	2.6	3.0	2.1	2.1	2.2

Table 5.8: Duration of translation in thousands of seconds for various values of *pop-limit* (columns) and *diversity* (rows) parameters.

5.4.3 Early Pruning

Some of the parameters in Moses prune the translation options before decoding starts. This type of pruning can potentially cause search errors. Unlike pruning during decoding, at load time, the decoder does not have access to target-side context (partial translation of the sentence so far).

Translation options for English words that map to several Czech variants with rich inflectional patterns can exceed some pruning threshold at load time. In this case, the decoder cannot distinguish between inflections that will not fit into translation context and those that will actually be needed when translating. The pruning is therefore based on insufficient information.

ttable-limit Defines how many alternative translations for a phrase are loaded from each phrase table. This parameter is automatically set in Moses configuration file by training scripts. Limit for the first table in one decoding path is set to 20 by default. For all subsequent phrase tables, the limit is set to 0 (unlimited). We evaluated various limit for the first table (30, 50, 100, 0), we kept the other steps at the default 0.

max-partial-trans-opt This option specifies the size of stacks during phrase expansion. The expansion is done before decoding, but it may explode as well: if there is a series of mapping steps that generate many expansions for each hypothesis, Moses has to prune these partial translation options *during* the expansion. Default value in Moses is 10000. We tried values 15000, 20000, 40000.

Parameter	Default
ttable-limit	30
max-partial-trans-opt	20000
max-trans-opt-per-coverage	10000
translation-option-threshold	0.0

Table 5.9: Early pruning parameters: default values.

ttable-limit	30	50	100	0
BLEU	13.12±0.45	13.37±0.46	13.23±0.45	13.26±0.47

Table 5.10: Early pruning parameters: ttable-limit.

max-trans-opt-per-coverage This option controls pruning before decoding but after the expansion of phrases. If pruning was necessary during the expansion, 10000 full hypotheses will have survived. These are pruned according to this limit, to 5000 per coverage by default. Our experiments cover values 10000, 15000 and 20000.

translation-option-threshold Similarly to beam search, two parameters modify pruning before decoding; the previous one is used for histogram pruning while this parameter controls threshold pruning. This is disabled by default. The values we evaluated were 0, 0.001, 0.0001.

We used Prospector in the "line exploration" mode to explore the possible settings—an exhaustive search in 4 dimensions would be too demanding. This algorithm tunes one variable at a time, the others are kept at their default (i.e. first) value. The default values are summarized in Table 5.9. This setting ensures that these variables will not create bottlenecks that would prevent other variables from being correctly evaluated. Tables 5.10, 5.11, 5.12 and 5.13 document our results. We ran the default setting 3 times and found almost no variation in BLEU: 13.12, 13.11, 13.12.

Translation table limit had a significant impact on the final BLEU score. Our experiments indicate that the default value is too low. The best result was achieved with the limit set to 50 options. Pushing the limit further resulted in a decrease of translation quality; 100 options performed similarly to the unlimited setting. We attribute this slight decrease to modeling errors. By allowing more translation candidates to be loaded, we might be making future pruning more difficult.

max-partial-trans-opt	15000	20000	40000
BLEU	13.44±0.48	13.12±0.45	12.28±0.44

Table 5.11: Early pruning parameters: max-partial-trans-opt.

By setting a larger maximum number of partial translation options, we allow the decoder to postpone pruning to a later stage when there is more information available. However this allows the intermediate steps to add very unlikely translation options, their combinations can explode in the following steps and they can make pruning more difficult (at this stage, pruning still occurs without sentence context). The decoder is forced to discard good candidates which results in a significant drop in BLEU score, as shown in Table 5.11.

max-trans-opt-per-coverage	10000	15000	20000
BLEU	13.12±0.45	13.40±0.46	13.37±0.45

Table 5.12: Early pruning parameters: max-trans-opt-per-coverage.

Increasing the maximum number of translation options per coverage lead to an improvement in BLEU (see Table 5.12. The evaluated system is apparently too complex for the default setting in Moses (5000). We achieved the best score with the value 15000, but the BLEU for 20000 translation options is only insignificantly lower; we are tempted to attribute this result to random variation during tuning, however more experiments would be needed for confirming this hypothesis.

translation-option-threshold	0.001	0.00001	0.0
BLEU	13.52±0.46	13.24±0.45	13.12±0.45

Table 5.13: Early pruning parameters: translation-option-threshold.

The results shown in Table 5.13 also seem paradoxical. By applying aggressive threshold pruning before decoding (i.e., before the decoder has complete information to decide on), we managed to increase BLEU score by 0.4 point absolute over the baseline system. It seems that there are modeling errors related to factored systems and that early pruning might help avoid generating bad translations.

The previous table shows that there are candidates to be found between the rank 10000 and 15000, here we observe that applying additional pruning to the first 10000 candidates significantly increases BLEU. There are some phrases that have many translation options, but these options are very unlikely compared to the best candidates. In these situations, threshold pruning makes the stacks smaller and probably helps to avoid search errors.

5.5 One Translation Step

Because of the difficulties discussed in the previous chapters—the absence of a reliable method for evaluation, the small and insignificant differences in BLEU and the enormous number of possible configurations—we did not carry out a fully automatic exploration of the space of factored setups. Instead, we used Prospector and the prediction of system complexity to simplify a manual search and conducted several sets of experiments in a few targeted research directions. We could call our experiments “semi-automatic”, mainly because Prospector was often used to evaluate various combinations of factors automatically. The achieved results are presented in the rest of this chapter.

The first set of experiments was done using Prospector's exhaustive search. We evaluated the usefulness of all additional factors in combination with the translation of surface forms. The setup was the following:

1. form|*extra* → form
2. (form → form)

All factors were evaluated with and without the alternative path (not back-off). Results are summarized in Table 5.14. Baseline system is denoted by '—'. The ± sign denotes the *standard deviation* over 3 runs of the optimizer. MERT was used for tuning the systems.

Factor	Single Path	+Alternative
—	9.93±0.03	—
afun	**10.08±0.08**	**10.11±0.09**
formeme	2.41±0.01	9.95±0.02
functor	9.08±0.08	**10.07±0.08**
gender	9.70±0.05	9.87±0.06
lemma	9.93±0.08	9.66±0.30
negation	**10.05±0.03**	9.99±0.02
number	10.00±0.03	9.96±0.08
person	9.92±0.03	9.79±0.18
sempos	9.93±0.06	9.95±0.16
tag	10.00±0.07	9.95±0.11
tense	**10.06±0.05**	**10.05±0.06**
tlemma	8.62±0.06	9.99±0.15
verbmod	9.56±0.04	9.94±0.10

Table 5.14: BLEU scores of configurations with 1 translation step.

We had to re-run even successful sets of MERT runs because for some factors, the optimization tended to diverge, resulting in deviations of several BLEU points and heavily skewed results. Some of the experiments required additional attention and manual repair work to finish. Overall, technical difficulties often hindered our progress and cannot be omitted among the reasons for the difficulty of automatic search for translation systems.

However, the results are highly interesting. First, the deviations are somewhat smaller than we report in Section 5.3. The fact that training, development and test data are from a single domain may have contributed to this outcome. While the initial experiments were carried out on random excerpts of CzEng 0.9, which is a mix of domains that often have quite different characteristics, all of the data in these experiments come from the news domain.

We still see only very little improvements over the baseline BLEU, complicated by variance that makes most of the differences insignificant. Even so, several factors stand out in both scenarios as potentially valuable for modeling the English-Czech translation.

Each column documents a different scenario and the results need to be interpreted accordingly. In the first column, factors that lead to data sparsity were penalized due to the absence of a back-off. Formeme stands out as the most prominent example, with the BLEU score 2.41 and almost no deviation; all MERT runs converged in a few iterations. Adding this factor diluted the data so much that translation became impossible. Factors that achieved high scores in this column can be (relatively) safely added to translation systems: they do not make the data much more sparse and increase translation quality. The best factors are highlighted in the table: analytical function, negation, tense. Grammatical number and tag are also potentially useful. If we only ran the experiments reported in the last column, we would not recognize the data sparsity issues of the other factors. They would be smoothed off by the alternative path.

In the second column, even factors that introduce some degree of data sparsity can achieve high scores—they may help in modeling some rare, but difficult phenomena. In the situations where the additional information is not helpful, the alternative path maintains good quality of translation. Functor, analytical function and tense appear to be the most promising factors according to this column.

5.5.1 Back-off Strategies

We chose three factors from Table 5.14 that performed best with the alternative decoding path and evaluated several options how to combine them and how to handle the trade-off between introduced data sparsity and complicated weight space.

Table 5.15 summarizes the results. Commas denote alternative decoding paths (not back-off, in the sense of Moses configuration). The first is the baseline system.

In the second system, we never joined all three factors, but let MERT balance the importance of each factor. The large variance points to the difficulties in optimization. In fact, no run of MERT converged, and all were stopped at the pre-defined limit of 25 iterations.

The third back-off strategy involved using all 3 factors jointly and backing off to translating form with the best factor. This setup was also not very stable. MERT runs converged, but showed considerable variance. The best run achieved BLEU 10.20 points.

In the next strategy, we joined the two best factors together for translation and backed this step off by the combination of form and the last (worst-performing) factor. We achieved insignificantly better BLEU score and slightly less variance.

Translation Steps	BLEU
form → form	9.93±0.03
form|afun → form : : form|functor → form : : form|tense → form	10.00±0.29
form|afun|functor|tense → form : : form|afun → form	10.08±0.10
form|afun|functor → form : : form|tense → form	10.10±0.08
form|afun|functor|tense → form : : form → form	**10.24±0.02**

Table 5.15: Back-off strategies and achieved BLEU scores.

The maximum achieved score was 10.22.

The last setup is the most straightforward, and the best according to any criteria. We simply joined all the additional factors together in a single translation step and used baseline form → form alternative path as back-off. This setup is entirely stable and achieves the best score. Note that we managed to improve translation quality noticeably just by using selected additional factors on the source side.

The complications and the need manual analysis in this section provide further empirical evidence that automatic search would have been impractical.

5.5.2 Additional Language Models

Factors that do not make the data too sparse (i.e., good factors in the first column in Table 5.14) can potentially be used to generate their counterparts on the target side. If these are informative as well, target-side language models trained on these factors can help guide the search. The factored setup in this case has no back-off, it is a joint translation of form|factor → form|factor.

Morphological tags have been used often in previous work, e.g. (Bojar et al., 2012b), in this setup. We evaluate other promising factors and report the results in this section (again, with 3 MERT runs and standard deviations).

Factor	BLEU
—	9.93±0.03
afun	10.03±0.03
negation	—
tense	10.03±0.02
tag	**10.83±0.07**

Table 5.16: BLEU scores with one additional language model.

Our results confirm the linguistic intuition and repeated reports of improvement when translating the morphological tags—in fact, we achieved a considerable improvement in BLEU score. For other factors, the additional language model had no benefit. We were unable to train a language model on the factor "negation" because it is almost always empty. SRILM smoothing failed on this data.

5.6 Multiple Mapping Steps

In this section, we evaluated a typical factored scenario with several factors. The scenario consists of two consecutive translation steps: lemma → lemma and one additional factor to its counterpart. This is followed by a generation step that takes the lemma and the additional factor and generates surface form on the target side. All of the factors have a language model on the target side. An alternative path maps surface form directly to all three target factors.

This setup has been used with tags in the past and improvements in BLEU have been reported. Our results are shown in Table 5.17. Systems without a score ran unbearably long (one MERT iteration took over a day). This is not surprising if we consider the prediction of the complexity. These experiments can also serve as additional evidence that our complexity estimation has practical use.

We achieved a large gain in BLEU (roughly 1.1 point absolute) when we used morphological tag as the additional factor, which confirms previous findings. It is disappointing that we were not able to find a novel configuration that achieves significantly better scores than our baseline system.

Factor	BLEU	Prediction of Complexity
—	9.93±0.03	7
formeme	9.91±0.05	4573
tag	**11.05±0.03**	655
functor	—	9903
sempos	—	38412
tense	—	13607

Table 5.17: BLEU scores of systems with 2 translation and 1 generation steps.

5.7 Discussion

5.7.1 Experimental Results

We were able to achieve improvement in BLEU over the baseline system in all the explored directions. We managed to improve the translation performance even when using a single translation step, merely by combining well-performing factors on the source side. We showed that analytical function, tense and functors as used in the PCEDT annotation are the most useful from a wide range of attributes for modeling transfer of English into Czech in this setting. We achieved an improvement of roughly 0.3 absolute BLEU points.

Furthermore, we showed that considering back-off strategies, the simplest approach—translating with the complex setup and alternatively form→form—gives the best results in terms of both the translation quality and the stability of weight optimization.

By generating an additional factor on the target side, we were able to increase the BLEU score by 1 point absolute over the baseline. However we could not find any other factors besides morphological tags that give this increase in translation quality.

We also evaluated a scenario that consists of multiple mapping steps. Unfortunately, similarly to the previous set of experiments, we were unable to identify any new useful factors, so even though our improvement in BLEU score is quite large (over 1.1 points), our findings are not original.

5.7.2 Search for Factored Configurations

The exploration of configurations discussed in this chapter was not automatic (we did create the experiments with various factors automatically using Prospector, but we designed the configuration of mapping steps, the number of factors etc. by hand). We have shown that factored models have some potential for improving machine translation. But it seems that finding the correct combination of steps and factors is not a task that an algorithm can solve, especially not by brute force—the number of possibilities explodes no matter which direction of exploration we take. A clever search in the space of configurations does not seem feasible due to the low reliability of automatic MT evaluation and frequent large variance in achieved scores across different optimization runs.

However we developed tools in this thesis that can assist us in carrying out the research and that allows us to abstract away from many details in configuration and factor selection.

5.7.3 Possible Research Directions

Regarding search for factored configurations, we believe it is possible but a particular research aim is necessary. The methods and tools provided in this thesis can assist in selecting the most suitable factored setup from a limited number of possibilities. Apart from using automatic metrics, we believe that unclear cases should be manually investigated, ideally using human evaluation tailored to the specific research goal (e.g., improving morphological coherence).

Recent work in analysis of machine translation research indicates that the most prominent problems of current methods may not lie in the task of statistical modeling.

In Daumé III and Munteanu (in review), the authors provide evidence that (for domain adaptation), bad hypothesis scoring is not a major issue, and neither are search errors (though the authors admit that they are difficult to measure). Instead, the main challenges lie in discovering new vocabulary items and new senses (translations) of known words.

These results correspond well with Turchi et al. (2009). This work contains a large-scale computational analysis of current problems in machine translation. The statistical model estimates are reportedly not the most significant problem. The authors suggest that instead, the focus of research should turn to obtaining relevant training data. Due to Zipf's Law, the positive effect of acquiring more data blindly rapidly decreases. Two possible approaches are proposed: adding training data by making targeted queries (active learning) and adding data generated by linguistic rules.

From this point of view, factored models as evaluated in this thesis are not very promising. Novel ways of exploiting this paradigm would have to be developed.

Adding new parallel data generated by rule-based systems as suggested above is one of the possibly promising directions for handling rich morphology in phrase-based machine translation.

Chapter 6

Conclusions

In this thesis, we provided a theoretical description of phrase based machine translation. We analyzed the paradigm of factored models and developed a method for estimating the complexity of factored configurations. We showed that while we are unable to predict the exact values, our estimate is nevertheless useful for research in this area.

We described the factors extracted from deep syntactic analysis of both the source and the target data. We provided a discussion of research related to our work.

We developed and presented a software tool called Prospector for automatic search in the space of possible configurations. Prospector supports several search algorithms, has a straightforward and extensible implementation and is well integrated into the current environment for machine translation experiments at our department. Its functionality proved invaluable for the work we carried out in this thesis.

We analyzed the difficulties associated with automatic exploration of factored configurations. We are not able to reliably recognize useful factors which prevents us from guiding our search in the space of possible setups. If we try to explore the space without this guidance, we are overwhelmed by the number of possible configurations, even in the simplest and most restricted settings. Furthermore, we provided an analysis of the number of back-off strategies, concluding that an automatic search for optimal back-offs is probably intractable. We were unable to overcome these problems and resorted to manually guiding the search, using the developed tools to automate the rest of the process.

In the experimental section, we evaluated a number of decoder options that modify pruning and search for translation hypotheses. This evaluation was simplified by the automatic search provided by Prospector. We carried out a large set of experiments in order to validate our evaluation methodology. We also briefly evaluated our attempt to include shallow syntax in the factored setting.

The focus of our experiments lied in the exploration of several research directions possible with factored models. We were able to improve the BLEU score but found that linguistic insight is irreplaceable when designing factored systems.

6.1 Future Work

We have not exhausted the possibilities of factored models by far in this work. Designing more experiments, especially in the direction of multiple mapping steps, seems like a promising topic for further work.

We would also like to continue adding new functionality to our tool Prospector, which we found quite useful in our research. Most importantly, we would like to develop additional search algorithms, such as various greedy search strategies.

Bibliography

Alexandra Birch and Miles Osborne. CCG supertags in factored statistical machine translation. In *In ACL Workshop on Statistical Machine Translation*, pages 9–16, 2007.

Ondřej Bojar and Zdeněk Žabokrtský. CzEng0.9: Large Parallel Treebank with Rich Annotation. *Prague Bulletin of Mathematical Linguistics*, 92, 2009. ISSN 0032-6585.

Ondřej Bojar, Kamil Kos, and David Mareček. Tackling Sparse Data Issue in Machine Translation Evaluation. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 86–91, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P10-2016`.

Ondřej Bojar, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. The Joy of Parallelism with CzEng 1.0. In *Proceedings of LREC2012*, Istanbul, Turkey, May 2012a. ELRA, European Language Resources Association. In print.

Ondřej Bojar. English-to-czech factored machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 232–239, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1626355.1626390`.

Ondřej Bojar, Bushra Jawaid, and Amir Kamran. Probes in a taxonomy of factored phrase-based models. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 253–260, Montréal, Canada, June 2012b. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-3130`.

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

C. Callison-Burch, P. Koehn, C. Monz, and O. Zaidan. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, page 22–64, Edinburgh, Scotland, 2011. Association for Computational Linguistics, Association for Computational Linguistics.

Daniel Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. Phrasal: A statistical machine translation toolkit for exploring new model features. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, pages 9–12, Los Angeles, California, June 2010a. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N10-2003`.

Daniel M. Cer, Christopher D. Manning, and Daniel Jurafsky. The best lexical metric for phrase-based statistical MT system optimization. In *HLT-NAACL*, pages 555–563. The Association for Computational Linguistics, 2010b. ISBN 978-1-932432-65-7. URL `http://www.aclweb.org/anthology/N10-1080`.

Mauro Cettolo, Marcello Federico, Daniele Pighin, and Nicola Bertoldi. Shallow-syntax phrase-based translation: Joint versus factored string-to-chunk models. October 21-25 2008. URL `http://www.mt-archive.info/AMTA-2008-Cettolo.pdf`.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999. URL `http://dx.doi.org/10.1006/csla.1999.0128`.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *ACL (Short Papers)*, pages 176–181. The Association for Computer Linguistics, 2011. ISBN 978-1-932432-88-6. URL `http://www.aclweb.org/anthology/P11-2031`.

Hal Daumé III and Dragos Munteanu. What's the problem with domain adaptation in machine translation. *Computational Linguistics*, in review.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of ACL*, 2010.

Qin Gao and Stephan Vogel. Consensus versus expertise: a case study of word alignment with mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, CSLDAMT '10, pages 30–34, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://portal.acm.org/citation.cfm?id=1866696.1866700`.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, Zdeněk Žabokrtský, and Magda Ševčíková-Razímová. Prague dependency treebank 2.0, 2006.

Kenneth Heafield. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W11-2123.pdf`.

Hieu Hoang, Alexandra Birch, Chris Callison-burch, Richard Zens, Rwth Aachen, Alexandra Constantin, Marcello Federico, Nicola Bertoldi, Chris Dyer, Brooke Cowan, Wade Shen, Christine Moran, and Ondřej Bojar. Moses: Open source toolkit for statistical machine translation, 2007. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.6165; http://www-csli.stanford.edu/~ccb/publications/moses-toolkit.pdf`.

Mark Hopkins and Jonathan May. Tuning as ranking. In *EMNLP*, pages 1352–1362. ACL, 2011. ISBN 978-1-937284-11-4. URL `http://www.aclweb.org/anthology/D11-1125`.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *ACL*. The Association for Computational Linguistics, 2007. URL `http://aclweb.org/anthology-new/P/P07/P07-1019.pdf`.

Howard Johnson, Joel D. Martin, George F. Foster, and Roland Kuhn. Improving translation quality by discarding most of the phrasetable. In *EMNLP-CoNLL*, pages 967–975. ACL, 2007. URL `http://www.aclweb.org/anthology/D07-1103`.

Philipp Koehn. Noun phrase translation. 2003. Adviser-Knight, Kevin.

Philipp Koehn. Statistical significance tests for machine translation evaluation, 2004.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521874157, 9780521874151.

Philipp Koehn and Hieu Hoang. Factored translation models. In *EMNLP-CoNLL*, pages 868–876. ACL, 2007. URL `http://www.aclweb.org/anthology/D07-1091`.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *HLT-NAACL*, 2003. URL `http://acl.ldc.upenn.edu/N/N03/N03-1017.pdf`.

Philipp Koehn, Amittai Axelrod, Ra Birch Mayne, Chris Callison-burch, Miles Osborne, and David Talbot. Edinburgh system description for the 2005 iwslt speech translation evaluation. In *In Proc. International Workshop on Spoken Language Translation (IWSLT*, 2005.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 135–139, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0x24`.

Jan Niehues and Alex Waibel. Domain adaptation in statistical machine translation using factored translation models. In *EAMT*, 2010.

Franz Josef Och. Minimum error rate training in statistical machine translation. In Erhard W. Hinrichs and Dan Roth, editors, *ACL*, pages 160–167. ACL, 2003. URL `http://aclweb.org/anthology-new/P/P03/#1000`.

Franz Josef Och and Hermann Ney. Improved statistical alignment models. In *ACL*. ACL, 2000. URL `http://aclweb.org/anthology-new/P/P00/`.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *ACL*, pages 311–318. ACL, 2002. URL `http://aclweb.org/anthology-new/P/P02/`.

Beatrice Santorini. *Part-of-Speech Tagging Guidelines for the Penn Treebank Project*. University of Pennsylvania, School of Engineering and Applied Science, Dept. of Computer and Information Science, Philadelphia, 1990. ISBN 0582291496.

Andreas Stolcke. SRILM - an extensible language modeling toolkit. In John H. L. Hansen and Bryan L. Pellom, editors, *INTERSPEECH*. ISCA, 2002. URL `http://www.isca-speech.org/archive/icslp_2002/i02_0901.html`.

Aleš Tamchyna and Ondřej Bojar. Bohatá anotace ve frázovém strojovém překladu. In *ITAT 2010 Informačné technológie – Aplikácie a Teória, Zborník príspevkov prezentovaných na konferencii ITAT*, pages 99–106, September 2010. ISBN 978-80-970179-3-4.

M. Turchi, T. De Bie, C. Goutte, and N. Cristianini. Learning to Translate: a statistical and computational analysis. Technical report, University of Bristol, 2009. URL `https://patterns.enm.bris.ac.uk/files/LearningCurveMain.pdf`.

Dániel Varga, László Németh, Péter Halácsy, András Kornai, Viktor Trón, and Viktor Nagy. *Parallel corpora for medium density languages*, pages 590–596. Benjamins, 2005. URL `http://www.kornai.com/Papers/ranlp05parallel.pdf`.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *COLING*, pages 836–841, 1996. URL `http://acl.ldc.upenn.edu/C/C96/C96-2141.pdf`.

Appendix A

User Manual for Prospector

A.1 Installation

Prospector can be downloaded as a part of the ÚFAL SMT playground from the following Subversion repository:

```
https://svn.ms.mff.cuni.cz/svn/statmt
```

Public access to this repository is possible using the username “public” and password “public”. Prospector assumes that it is run inside the environment of a “playground”. It is not a standalone application and does not function otherwise.

A.2 Configuration

In order to run Prospector, the user has to create a configuration directory containing the following files:

- rules
- vars
- traceback
- score

The file `score` is an executable that produces a number corresponding to the score or “fitness” of a given final eman step.

Apart from these files, the directory also has to contain a subdirectory `chunks` with parts of eman tracebacks.

A.2.1 Traceback

This file defines how the individual chunks should be combined:

```
eval
  mert
    tm
      align
    lm
```

The lines correspond to names of files in the directory `chunks` and indentation defines which steps depend on each other. Multiple eman steps can be defined in one file. This file exists to add flexibility to configuration. In the future, we would like to extend it to allow alternative/conditional definitions.

The chunks can contain *variable slots* surrounded by '#' sign:

```
+- s.align.e357fb70.20120221-1115
|  | ALILABEL=en-#SRCFACTOR#-cs-#TGTFACTOR#
|  | ALISYMS=gdfa
|  | CORPUS=czeng-news
|  | GIZASTEP=s.mosesgiza.fcfbe812.20120221-1114
|  | SRCALIAUG=en+#SRCFACTOR#
|  | TGTALIAUG=cs+#TGTFACTOR#
```

A.2.2 Variables

Each line of this file defines a variable and its values. The first column is the variable name (must match the slot name in the traceback) and the second column (separated by any number of spaces) contains possible values for that variable, separated by commas.

A.2.3 Rules

This file places restrictions on possible combinations of variable values. The user can optionally define these to avoid evaluating nonsensical configurations or to direct the search. It contains lines with 4 space-delimited columns, for example:

```
TMSRCAUG   /\+/        STEPS   t0a1-0
TMSRCAUG   /^[^+]*$/   STEPS   t0-0
```

Each line can be viewed as an if-statement: *if* 1 matches 2 *then* 3 must equal 4. Numbers represent the columns on the line. The second column is evaluated as a Perl regular expression. The fourth column must contain an exact value.

A.2.4 Prediction

The user can optionally also include a file `predict`. It has to be an executable that will, given the path to the final step, output the complexity or cost estimate for such an experiment. If the user also specifies a limit, Prospector will query this program before running each experiment and it will discard experiments with cost over the threshold.

A.3 Running Prospector

Usage:

```
prospector [options] config-directory
```

A.3.1 Command-Line Arguments

max-running Maximum number of experiments running in parallel.

search Search type. Possible values:

- genetic
- exhaustive
- line
- random

genetic-population-size Size of one generation in genetic search.

genetic-nbest The number of best configurations considered as parents for next generation.

genetic-mutation-prob Probability of mutation in genetic search.

random-limit The total number of experiments created in random search.

max-allowed-prediction Threshold value of prediction.

verbose Be verbose.