3.4: Evaluation of Tree Transfer System

Ondřej Bojar, Miroslav Týnovský

Distribution: Public

EuroMatrix
Statistical and Hybrid Machine Translation
Between All European Languages
IST 034291 Deliverable 3.4

March, 2009

Project funded by the European Community
under the Sixth Framework Programme for
Research and Technological Development.

Project ref no.	IST-034291
Project acronym	EuroMatrix
Project full title	Statistical and Hybrid Machine Translation Between All European Languages
Instrument	STREP
Thematic Priority	Information Society Technologies
Start date / duration	01 September 2006 / 30 Months

Distribution	Public
Contractual date of delivery	February, 2009
Actual date of delivery	March, 2009
Deliverable number	3.4
Deliverable title	Evaluation of Tree Transfer System
Type	
Status & version	
Number of pages	29
Contributing WP(s)	3
WP / Task responsible	Jan Hajič
Other contributors	
Author(s)	Ondřej Bojar, Miroslav Týnovský
EC project officer	Xavier Gros
Keywords	

The partners in EuroMatrix are:
Saarland University (USAAR)
University of Edinburgh (UEDIN)
Charles University (CUNI-MFF)
CELCT
GROUP Technologies
MorphoLogic

For copies of reports, updates on project activities and other EuroMatrix-related information, contact:
The EuroMatrix Project Co-ordinator
Prof. Hans Uszkoreit
Universität des Saarlandes, Computerlinguistik
Postfach 15 11 50
66041 Saarbrücken, Germany
uszkoreit@coli.uni-sb.de
Phone +49 (681) 302-4115- Fax +49 (681) 302-4700

Copies of reports and other material can also be accessed via the project's homepage:
http://www.euromatrix.net/

Contents

Chapter 1

Introduction

This report presents our progress and final status of tree-based transfer for machine translation, covering thus the aims of Work Package 3 of the EuroMatrix project. Further details and other Deliverables are available at our departmental web site[1] or the official EuroMatrix web site[2].

Chapter 2 briefly describes the progress in manual annotation of parallel texts at Czech and English tectogrammatical layers (see Deliverable 3.1, Mikulová et al. (2007)) and also our parallelization of automatic annotation pipeline.

Chapter 3 is devoted to further improvements and empirical evaluation of the Tree Aligner proposed in Deliverable 3.2 (Bojar and Čmejrek, 2007) and implemented in Deliverable 3.3 (Bojar et al., 2008b).

The key Chapter 4 evaluates our tree-to-tree transfer system TreeDecode. We first examine some problems specific to the transfer at the tectogrammatical layer and continue with various improvements of TreeDecode implementation and an empirical evaluation of various setups. Chapter 4 concludes with a comparison to our configuration of a state-of-the-art phrase-based MT system Moses and other English-to-Czech MT system that took part in the shared translation task of EACL 2009 Fourth Workshop on Statistical Machine Translation (WMT09[3]).

A separate Chapter 5 examines the utility of tectogrammatical layer for automatic evaluation of MT quality. We observe, that a novel metric SemPOS based on some features from the tectogrammatical layer performs best in terms of correlation with human judgments.

The final Chapter 6 summarizes our observations and achievements.

[1] http://ufal.mff.cuni.cz/euromatrix/

[2] http://www.euromatrix.net/

[3] http://www.statmt.org/wmt09/

Chapter 2

Data Annotation and Pre-Processing

2.1 Progress in Manual Annotation at the t-Layer

One of our tasks in Work Package 3 was to design a formal representation of sentence structure and meaning, see Mikulová et al. (2007). We have extended our aims and we are in the process of manual annotation of English sentences from Prague Czech-English Dependency Treebank (PCEDT, Čmejrek et al. (2004)) at the t-layer. With funding support from additional sources, the Czech sentences in PCEDT are also being manually annotated following the representation designed for the Prague Dependency Treebank (PDT, Hajič (2004)).

While the two annotation processes are not completely synchronized, the annotators aimed at a moderate overlap. Table 2.1 documents the current state of annotation: sentences with manual annotation at the t-layer finished in both languages.

	English	Czech
Sentences	4599	
Manual Annotation		
a-nodes (i.e. tokens)	115109	107432
t-nodes	78004	86973
Automatic Annotation		
a-nodes (i.e. tokens)	111290	107787
t-nodes	70988	75875

Table 2.1: Current size of manually annotated Czech and English sentences in PCEDT. Only finished and revised sentences in both languages are included.

We see that the PCEDT dataset contains rather long sentences, 23 Czech and 25 English tokens per sentence on average and that the annotation to t-layer leads to the reduction of node count by factor of nearly 1.5 for English and 1.2 for Czech. There is an important difference between automatic and manual t-layer: the manual annotation closely follows the guidelines and includes "generated" nodes, i.e. nodes with no corresponding word in the surface representation of the sentence but required in the tectogrammatical valency frame (e.g. elided inner participants such as dropped pronouns in subject position). Automatic procedures do not use any valency dictionary and they are therefore not capable of adding that many elements.

2.2 Parallel Large-Scale Rich Automatic Pre-Processing

Being aware of the fact that manually annotated data are and will be inevitably too small to train transfer models of large coverage, we are developing and improving tools for automatic annotation. Our pipeline for automatic annotation was described in Section 4.1 of Deliverable 3.2 (Bojar and Čmejrek, 2007).

Since then, we have moved from our own pipeline setup to the generic framework of TectoMT (Žabokrtský et al., 2008; Žabokrtský and Bojar, 2008) that is being implemented. TectoMT includes many NLP tools, provides a unified file format and a very effective means of building complex annotation pipelines, so called **scenarios**.

Our contribution to TectoMT is a tool for running scenarios on large datasets in parallel on a Sun Grid Engine cluster. Thanks to the parallelization, we were able to annotate large Czech monolingual and Czech-English parallel data automatically up to the t-layer. Table 2.2 shows the magnitude of data we were able to process on our cluster of 40 4-CPU machines in a few weeks.

Czech Monolingual Data

Sentences	52 M
with nonempty t-layer	51 M
a-nodes (i.e. tokens)	0.9 G
t-nodes	0.6 G

Czech-English Parallel Data

	English	Czech
Sentences	6.91 M	
with nonempty t-layer	6.89 M	
a-nodes (i.e. tokens)	61 M	50 M
t-nodes	41 M	33 M

Table 2.2: Czech monolingual and Czech-English parallel data annotated automatically up to the t-layer.

The data serves our experiments with transfer at the t-layer and was also used in the WMT09 shared task. See Bojar et al. (2009) for more details on data sources and applications.

Chapter 3

Tree Aligner

3.1 Improvements of the Perl Implementation

The original implementation of our tree aligner in Perl as described in Deliverable 3.3 (Bojar et al., 2008b) poses some problems when applied to larger data or utilized in connection with other tools:

- The implementation requires Perl modules Data::Dumper or Storable installed and produces models stored in structures readable only using these modules. In order to employ the learned models in other tools including our Tree Decoder (see Section 3 of Deliverable 3.3), we need a portable file format.
- Each iteration builds the whole new model in memory. For larger datasets, the model cannot fit into computing memory (see Table 2.7 in Deliverable 3.3).
- In each iteration, the whole scoring model is loaded into memory. Loading only individual rules on demand can greatly reduce memory usage at runtime.

These problems relate to the way of building and storing the model files. Proposed and implemented solutions are described in Sections 3.1.1 and 3.1.2.

Another problem is the time consumption of the aligner. We adapted the software to collect (weighted) observation counts on disjoint sets of sentence pairs in parallel. The new model is created by joining the independent observations and converting counts to probability estimates as usual. The parallelization is described in Section 3.1.4.

The pruning methods were slightly improved, too. These improvements are described in Section 3.1.6. We have also generalized the input file format (Section 3.1.7) allowing backoff models to choose from more input attributes.

3.1.1 Models File Format for Interchange

The models are now stored in simple text files which can be used by the decoder or any other tool. Using this file format also avoids the dependency on external Perl modules.

The model file format is line-oriented. Each line represents one rule and its probability. More precisely, there are seven tab-separated fields on each line:

- backoff model index
- state of the left treelet root
- state of the right treelet root
- left treelet structure, internal node labels and frontier node states
- right treelet structure, internal node labels and frontier node states

- pairing (alignment) of frontier nodes
- conditional probability of the rule (the pair of treelets with aligned frontier nodes given their root states)

When exporting the model for another tool such as the decoder, we store the whole model in a single file. During the computation, we split model file into parts, so called "submodels". A single submodel file contains the set of rules sharing the backoff model type and the root state (of both left and right treelets). As all the conditional probabilities in each submodel file share the same condition (the root state), this division matches the formula of conditional probability and the probabilities in one file thus sum to one. The following section shows how we use this formula while building a model.

3.1.2 Improved Model Estimation

The key difference between the former and the current implementation is the division of model estimation into two phases.

The first phase (see Alg. 1 and compare to Alg. 3 in Section 2.2 of Deliverable 3.3) just goes through the data, collects all observations of each rule and computes the probability of using the rule. The pairs of the observed rule and the observation probability are stored in several files. Each file contains rules with the same root state (i.e. the left hand side).

The second phase (see Alg. 2) adds up the collected observation probabilities for each rule to get its new expected count. The new probability of a rule is then computed as a division of the rule's expected count and the sum of expected counts of all rules with the same root state.

Note that these rules are stored in the same file, therefore the sum of their expected counts is equal to the sum of all observation probabilities in the file. The second phase can hence process each file separately. The result is a set of submodel files that form the whole new model.

This two-phase setup saves the required memory in several ways: in the first phase, we need access to the scoring model (the model from the previous iteration) to calculate rule observation probabilities. As soon as a rule observation probability is calculated, it is written to the disk and not needed until the second phase. So in the first phase, we're saving by not needing the new model. In the second phase, the scoring model is not needed any more: we just add up observation counts. Moreover, thanks to the independence of submodel files, we can run the second phase for all submodels sequentially (or in parallel on several machines) and thus work with a fraction of the overall model data at a time.

Algorithm 1 Collecting observation probabilities.

```
1.  for each (left_tree, right_tree) ← tree-pair from input file
2.      Rs ← observe synchronous rules(left_tree, right_tree)
3.      for each rule R∈Rs in safe order
4.          count inside probability update of R for the chart cell
5.              determined by left and right treelet bottom up indices
6.      end
7.      for each rule R∈Rs in reversed safe order
8.          count outside probability update of R for the chart cell
9.              determined by left and right treelet bottom up indices
10.     end
11.     for each rule R∈Rs
12.         P ← observation probability (i.e. expected count update of R)
13.         S ← root state (left hand side) of R
14.         append the tuple (R, P) to the file called S
15.     end
16. end
```

Algorithm 2 Computing rule probabilities from observations.

```
1.  for each rule
2.      expected_counts(rule) ← 0
3.  end
4.  for each F ← file of collected observations
5.      total count ← 0
6.      while (rule R, observation probability P) ←  read F
7.          increment expected counts of R by P
8.          increment total count by P
9.      end
10.     for each affected rule R
11.         probabilities(R) ← expected count of R / total count
12.     end
13. end
```

3.1.3 Model File Format for Re-Estimation

For the purposes of re-estimation of model parameters as described in Section 3.1.2, the submodels are stored in CDB database file format instead of the interchange plain text format (Section 3.1.1). This enables additional memory savings because the next iteration does not have to to load the whole scoring model. The rule probabilities are loaded on demand as needed for the computation.

We use the Perl module `CDB_File` to read and write the CDB file format. The module also ensures minimalization of disk access by reasonable caching of those rule probabilities which are used often.

3.1.4 Parallelization

Aside from memory savings, the two-phase model estimation also enables time savings in form of parallelization of the process. The parallelization is possible due to the fact that the expected count updates can be collected independently for each tree pair. Therefore we can split the input data into disjoint sets and run the first update-collecting phase on each set separately, in parallel. When all the parallel processes are finished, we merge the corresponding submodel files and perform the second phase as usual. Note that the second phase can also be parallelized by running it for each submodel file independently.

The parallelization uses Sun Grid Engine cluster.

3.1.5 Improved Detaching Lexical Information

Another minor memory savings are acquired by an improvement of detaching lexical information (see Section 2.4.1 of Deliverable 3.3). The frequencies of the detached lexica (lemmas, functors, etc.) are now considered. Therefore the lower, less memory consuming indices are assigned to the more frequent lexica.

3.1.6 Improved Pruning Methods

In Deliverable 3.3, we proposed several improvements of pruning methods. Two of them are implemented in the current version:

InternalsReflectWordAlignmentBenevolent. A benevolent variant of pruning treelet pairs with not aligned internal nodes. This method requires only one pair of the internals word-aligned in order to keep a treelet pair.

PruneLowFrequencyBackoffTreelet. In the pruning of low frequency single treelets we can now specify the level of detail considered: the chosen back-off model determines which of the treelet features (structure, frontier states and internal labels) are preserved when

calculating treelet frequency. A rule is not created if either the left or the right treelet (in the specified coarse representation) were observed too rarely.

3.1.7 Generalized Input File Format

The input file format is more general. It is very similar to the file format of the former implementation and it is compatible with it. The only difference is that a token can include variable number of lexical factors instead of the fixed number of three.

The last three factors contain the word-alignment and structure information, this remains unchanged.

The extension of the input file format is complemented by the extended configuration options of backoff models (Section 3.1.8) that allows to choose any of the factors as node labels.

3.1.8 Experiment Configuration

We use a single configuration file to store the whole configuration of an experiment. As the computation runs in parallel, manual execution of the iterations would be hard to manage. Currently, all iterations are executed by a single script which gets the configuration file as an argument. The configuration file was enriched with options describing the iteration number to start and end with and the input data to use. We also added options for the new pruning methods.

The following list contains all possible options and their values in a configuration file:

`backoffs` Set of space-separated backoff definitions. Each backoff definition has a form of `a-b-c-d`, where

- `a` is a set of indices which determine frontier node state,
- `b` is a set of indices which determine internal node label,
- `c` is a boolean (1 or 0) value determining whether treelet structure is preserved and
- `d` is a weight of the backoff model.

For example `backoffs = 1-0-1-4 2-1-0-1` defines two backoff models. The first one uses the second factor (semantic part of speech on Czech side and functor on English side) as frontier node state, the first factor (t-lemma) as internal node label and keeps the treelet structure. Its relative weight is 4 (the absolute weight is 0.8).

The second backoff model uses the third factor (formeme) as frontier node state, the second factor as internal node label and omits the treelet structure. Its relative weight is 1 (the absolute weight is 0.2).

The specific meanings of the factors naturally depends on the factors in the training data file.

`left_transcode_dicts`, `right_transcode_dicts` Space separated list of dictionaries of lexica (words) used in the original input trees. The dictionaries are stored in very simple format; a single word is listed on a single line. The more frequent words come earlier. Each word is replaced with its line number in the data with the detached lexical information.

`left_treelet_pruning`, `right_treelet_pruning` Which treelet pruning method is used for the left and right treelets in each rule. Possible values:

- `no_pruning`: no pruning method is used, all observed treelets are kept.
- `is_treelet_hifreq`: only the treelets contained in the set of high frequency treelets are kept.

`left_treelet_counts_filename`, `right_treelet_counts_filename` Names of files containing precomputed counts of left and right treelets in the input data. These counts are used in `is_treelet_hifreq` pruning method.

`hifreq_pruning_bm_index` This option enables the new feature described in Section 3.1.6. It determines which backoff model is used for checking whether a treelet is frequent enough not to be pruned by `is_treelet_hifreq` method.

`hifreq_pruning_treshold` How many times the treelet must be observed in the data not to pruned by `is_treelet_hifreq` method. The value of 1 is equivalent to no pruning, the default value of 2 is the most benevolent variant that does prune some treelets.

`treelet_pairs_pruning` Which treelet pair pruning method is used. Possible values:

- `no_pruning`: All treelets are kept
- `do_treelets_reflect_wali`: A treelet pair (t_1, t_2) is pruned if there exists a word-alignment edge (a, b) where a is a node inside t_1 and b is a node outside of t_2 or vice versa, i.e. the edge joins a node inside one treelet and a node outside of the opposite treelet.
- `do_internals_reflect_wali`: A treelet pair (t_1, t_2) is pruned if there exists a word-alignment edge (a, b) where a is an internal node inside t_1 and b is a node outside t_2 or vice versa, i.e. the edge joins an internal node of one treelet and a node outside the opposite treelet.
- `do_internals_reflect_wali_benevolent`: A treelet pair is pruned if all word-alignment edges, which start from an internal node of one treelet lead to a node outside the opposite treelet.

`rule_pruning` Which rule pruning method is used. Possible values:

- `no_pruning`: All rules are kept.
- `do_frontiers_reflect_wali`: A rule is pruned if it contains a frontier-mapping edge (a, b) and there exists a word-alignment edge (a, c) where $c \neq b$.
- `do_frontiers_reflect_wali_benevolent`: A rule is pruned if it contains a frontier-mapping edge (a, b) and there exists a word-alignment edge (a, c) where c lies outside the subtree of b.

`data_files` A set of space-separated paths to input files of tree pairs. Within a single iteration, each file is processed separately independently to the others. This option thus determines the level of parallelization. The number of files is also the number of processes which will be executed in parallel.

`start_with_iteration` What iteration to start with. Any number higher than 0 assumes the existence of previous iteration model. The default value is 0.

`end_with_iteration` What iteration to end with. The default value is 1. The computation of following iterations can be taken up again using the `start_with_iteration` option and re-executing the experiment.

`experiment_id` Identification string of an experiment. It is used as a name of subdirectory where the models, log and additional files are stored.

`virtual_memory_size` The limit of the amount of operating memory that each process can use. It is specified in megabytes. The default value is 512 (MB).

`model_prefix` The prefix of each model subdirectory of the experiment directory. The name is completed with the number of iteration. The submodels of each iteration are stored into the subdirectory. The default value is `model_`

3.1.9 Experiment Execution

As explained before, two phases of computation are executed in parallel:

- computing the new model parts
- merging them together

Alg. 3 shows the procedure of executing parallel processes.

Algorithm 3 Running the aligner in parallel.

```
1.  dataset D ← read the set of data files from configuration file
2.  start ← read the number of iteration to start with
3.  end ← read the number of iteration to end with
4.  for i ← start to end
5.      for each part P∈D
6.          run i-th iteration with input P on a machine
7.          add affected submodels to a set SUBS of submodels
8.      end
9.      wait until all the processes finish
10.     for each submodel S∈SUBS
11.         run merging of the computed parts of S on a machine
12.     end
13.     wait until all the processes finish
14.     /* the set of merged submodels forms the new model */
15. end
```

3.2 Evaluation

In this section, we describe experiments testing the performance of the aligner depending on the backoff models, pruning methods and training data used.

The experiments are divided into three parts:

- Section 3.2.1 documents the speedup of the current implementation compared to the previous version (Deliverable 3.3).
- Section 3.2.2 reports the number of rules extracted and the coverage of the ruleset depending on the backoff models and pruning methods.
- Section 3.2.3 compares the performance of the aligner for manual or automatic input trees.

3.2.1 Speedup of the Current Implementation

Table 3.1 compares the amount of computation time of the "zeroth" and the first subsequent EM iteration on one hundred tree pairs using the current parallel version and the former non-parallel version. The comparison is not fair because the parallel version uses 10 machines to get the results. The comparison is useful to estimate the the costs of parallelization itself, we can see that the parallel version is not 10 times better but only about 7 times. However, the larger data are used for computation, the lower impact the costs have.

	"zeroth" iteration	EM iteration
parallel (current) version	22 s	26 s
non-parallel (previous) version	148 s	186 s

Table 3.1: Comparison of time consumption of parallel and non-parallel version

3.2.2 Comparison of Different Backoff Models and Pruning Methods

In this section, we compare the coverage of the input sentence pairs by extracted rules depending on which backoff model is taken into account and what pruning method we use.

There are a lot of possible settings as a pruning method and a backoff model are set independently. The goal is to cover as many tree pairs as possible and to keep the model reasonably small in order to keep the memory and time consumption bearable. A pruning method decreases the model size by not taking some rules into account whereas a backoff model does it by decreasing the data sparseness.

For the comparison, we use the Project Syndicate section of CzEng (Bojar et al., 2008a)[1]. The number of sentence pairs is 84,141. Omitting sentences which contain a node with more than five children, the number of sentences sinks to 75,335. Of all the available attributes at the t-layer (Mikulová et al., 2007), we use only t-lemma, functor, formeme and semantic part of speech. Most of our pruning methods consider node-to-node alignment when selecting likely treelet pairs. We use the automatic alignment by Mareček et al. (2008) which is in turn based on GIZA++ "word" alignments of linearized trees.

We do not list all possible combinations of the pruning methods and backoff models. Table 3.2 shows several possible settings of pruning methods with the backoff model fixed to 'label = t-lemma, state = functor, structure = yes' (for further combinations see Deliverable 3.3, Table 2.7). Table 3.3 shows several possible settings of backoff models with the pruning method fixed to 'internals reflect wali benevolent + frontiers reflect wali benevolent'.

Pruning method	Number of rules	Tree pairs covered	Tree pairs covered (%)
treelets reflect wali	$28.2 \cdot 10^6$	4,942	6,56 %
internals reflect wali benevolent + frontiers reflect wali benevolent	$40.8 \cdot 10^6$	18,881	25.06 %

Table 3.2: Performance of the aligner depending on chosen pruning method.

Backoff model	Number of rules	Tree pairs covered	Tree pairs covered (%)
label: t-lemma, state: functor, structure: yes	$40.8 \cdot 10^6$	18,881	25.06 %
label: sempos, state: formeme, structure: yes	$34.4 \cdot 10^6$	18,896	25.08 %
label: sempos, state: functor, structure: yes	$35.8 \cdot 10^6$	18,891	25.08 %

Table 3.3: Performance of the aligner depending on chosen backoff model.

We see that the pruning method "treelets reflect wali" is too strict. Even with 28 million rules extracted from the 75k sentence pairs, only about 5k sentences can be reconstructed (see the column "Tree pairs covered") in Table 3.2. The more benevolent method increases the percentage of reconstructable trees to 25 %.

Table 3.3 shows statistics of the model depending upon the backoff model used. The decreasing counts of observed number of rules indicate more and more general backoff models (with a lower vocabulary size of node labels and frontier states). Still, the number of tree pairs that can be reconstructed by the learned rules ("tree pairs covered") is not any higher for more general models: the used pruning method (internals reflect wali benevolent + frontiers reflect wali benevolent) is nearly equally destructive for all the backoff models.

To make the aligner usable in practice, some sort of generalization of tree structure is necessary, such as learning, which dependents are "obligatory" and which are "free modifiers". The rules should then map only obligatory elements to obligatory elements, and all "free modifiers" would be mapped on the fly.

[1] `http://ufal.mff.cuni.cz/czeng/`

3.2.3 Comparison of Manually and Automatically Annotated Data

This set of experiments uses the data described in Table 2.1. Both the manually and the automatically annotated data were enriched with automatic alignment of t-nodes and exported to the input format of the aligner.

Table 3.4 shows the numbers extracted rules and the percentage of covered trees for annotations. We use the pruning method "internals reflect wali benevolent + frontiers reflect wali benevolent" and the backoff model "label=sempos, state=functor, structure=yes".

Type of annotation	Number of rules	Tree pairs covered	Tree pairs covered (%)
automatic	$1.79 \cdot 10^6$	411	10.5 %
manual	$1.86 \cdot 10^6$	437	11.1 %

Table 3.4: Performance of the aligner depending on whether automatic or manual annotation is used.

We see that with this different dataset, only 10 to 11 % of tree pairs can be reconstructed using the extracted rules. We attribute the effect to the different domain of texts.[2]

At least some positive news is the minor improvement in coverage when using manual trees instead of automatic ones. This confirms that the tectogrammatical layer is designed reasonably and improving both sides (manual annotations are surely closer to the definition of the layer than automatic parses are) indeed increases the structural similarity, the key motivation for the whole enterprise.

[2]The lower coverage cannot be explained by less training data, because we are testing the coverage *on the training data*, not on an independent test set.

Chapter 4

Tree Decoder: TreeDecode

The tree-to-tree transfer system TreeDecode (Deliverable 3.3, Chapter 3, Bojar et al. (2008b)) is a highly configurable system that translates source dependency trees to target dependency trees. The system can be applied at or across various layers of linguistic representation of the sentence, as long as the representation is a dependency tree. Figure 4.1 illustrates the possible setups we are interested in when translating from English to Czech: "etct" is our main goal, transfer at the tectogrammatical layer, "eaca" is the application of the system at surface-syntactic (analytical) layer and "epcp" is an approximation of phrase-based translation: the source and target "trees" are linear trees, branching to the right hand side only.

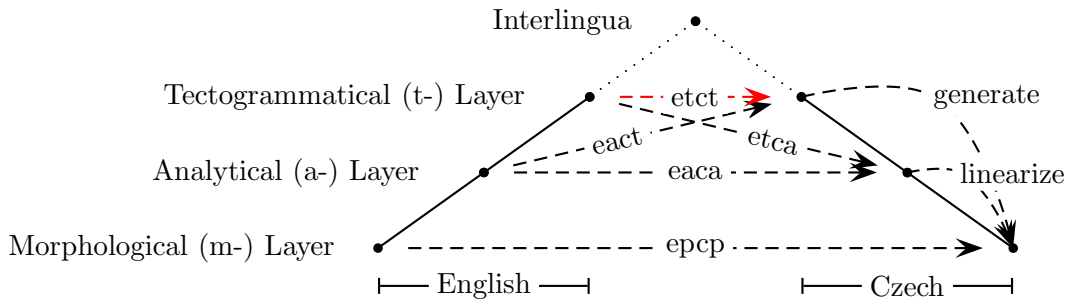

Figure 4.1: Various setups of tree-to-tree transfer.

The main motivation for tree-to-tree transfer is its application at the t-layer (Mikulová et al., 2007), the deep syntactic representation of a sentence.

Unless stated otherwise, the scores reported in this text are based on the WMT09 development set, part b, containing 1026 sentences from various news web sites. We report BLEU (Papineni et al., 2002) with empirical confidence intervals at 95% level estimated using bootstrapping (Koehn, 2004).

4.1 Considerations on the Complexity of Deep Transfer

This section describes some specific properties of the t-layer relevant for statistical MT transfer, the new problems caused and proposals to solve them.

4.1.1 Upper Bound given Czech Target Generation

When translating via t-layer, the last step in the translation pipeline has to be the generation of the target string of words. Naturally, the quality of the output is bounded by the quality of this generation step.

To estimate the upper bound on MT quality given the generation step, we performed a simple experiment, see Figure 4.2. We use our automatic analysis of Czech target sentences

(to approximate the Czech t-trees we see in the training phase) and re-generate them using the deterministic generation sequence implemented in TectoMT (Žabokrtský et al., 2008). The English source does not come into play at all.

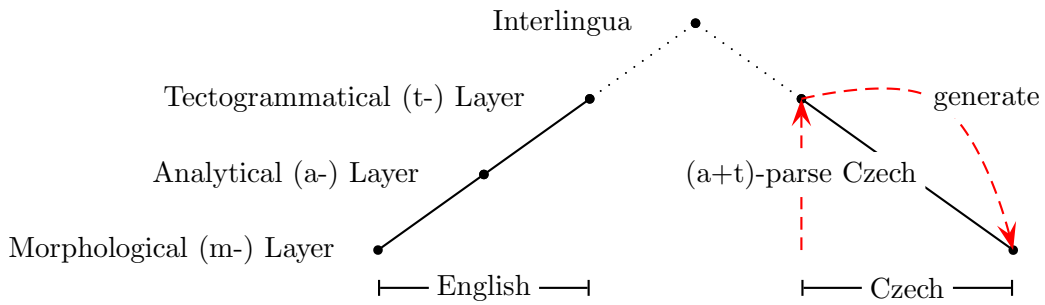

Figure 4.2: Experimental setup for upper bound estimate of BLEU given Czech target generation.

Table 4.1 presents BLEU scores from this Czech-to-Czech "translation" and documents the importance of attributes of t-nodes. In our transfer, we use a new attribute "formeme" introduced by Žabokrtský et al. (2008) and the most important 23[1] of all the 38 content attributes listed in the Appendix of Deliverable 3.1.

We compare BLEU scores for the setting where all these 24 attributes are preserved to situations when some of the attributes are deleted and the generation procedure has to use some default instead of the original value. With all attributes reserved, the score of 57 % BLEU can be interpreted as an upper bound on BLEU for any English-to-Czech system that finishes with this generation step.

Ignoring all grammatemes (the "gram/*" attributes), leads to a loss of nearly a half of the BLEU score. Ignoring all attributes except for t-lemma reduces BLEU to nearly one sixth. This difference indicates how important the contribution of all various morphological markings of content words to the meaning of the sentence is.

Rather interesting is the impact of removing formeme and preserving all other attributes: BLEU sinks from 57 to 14 points. The formeme proves to be the single most important attribute for the generation procedure in TectoMT. For this reason and also because of the definition of formeme (to express the syntactic relation between the governing and the dependent node), we choose formeme as the state of frontier nodes.

Attributes of t-nodes preserved	BLEU
All attributes preserved	57.09±1.14
Preserved t-lemma, functor, nodetype, formeme, but no grammatemes	33.43±1.02
Preserved all attributes except for formeme	13.87±0.70
Preserved only t-lemma, functor and nodetype	11.32±0.66
Preserved only t-lemma	11.11±0.67

Table 4.1: BLEU scores for Czech-to-Czech "translation" via t-layer.

[1]Proper t-layer attributes used in our transfer: t-lemma, functor, nodetype, gram/sempos, gram/number, gram/negation, gram/tense, gram/verbmod, gram/deontmod, gram/indeftype, gram/aspect, gram/numertype, gram/degcmp, gram/dispmod, gram/gender, gram/iterativeness, gram/person, gram/politeness, gram/resultative, is_passive, is_member, is_clause_head, is_relclause_head.

4.1.2 Cannot Assume Atomicity of t-nodes

The basic model of Synchronous Tree-Substitution Grammars (STSG), as specified in Deliverable 3.2 (Bojar and Čmejrek, 2007), assumes only two pieces of information at a node: the (internal) node label and the (frontier and root) state. Put differently, STSG assumes that node labels are atomic, there is no means to work with individual attributes, except using some of them as frontier states.

Section 4.1.1 documented the importance of t-node attributes: in order to hope for a reasonable BLEU score, we have to provide most of the attributes. To accept the atomicity assumption, we have to concatenate all t-node attributes of a node to a single compound value. This naturally breaks one of the main advantages of t-layer, the independence of various attributes in translation: the source-language number or tense marking can be mapped to the target-language number or tense marking nearly regardless the lexical value (the t-lemma) of the sentence element in question. And another cost we have to pay for the atomicity assumption is the increased vocabulary size: the vocabulary size can in theory grow up to the full cartesian product of t-node attributes. In practice, not all attribute combinations are meaningful.

Figure 4.3 examines, whether we can find a reasonable balance of the vocabulary complexity and achievable BLEU score assuming atomic node labels.[2]

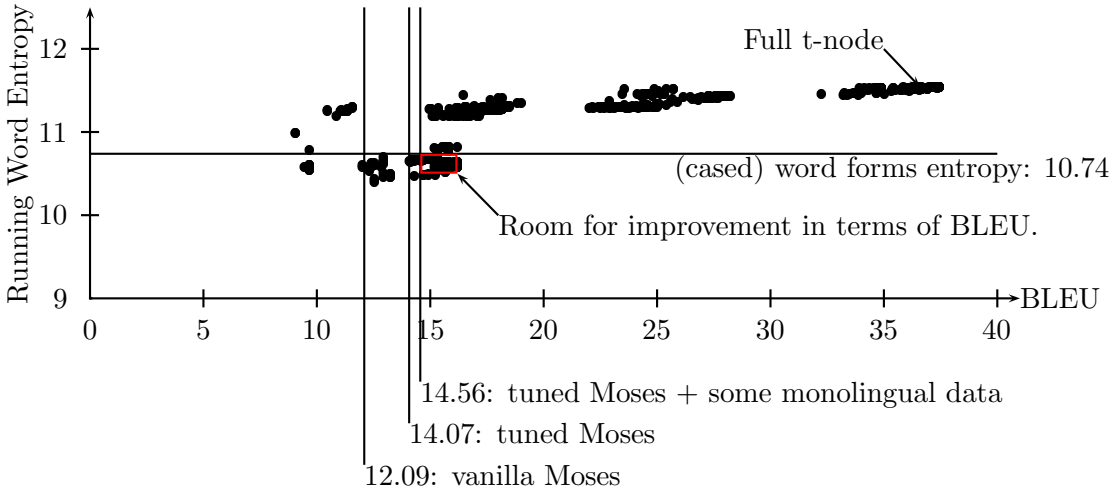

Figure 4.3: Room for improvement in BLEU score assuming atomicity of t-nodes and the goal to keep the entropy of the model below entropy of plaintext.

Each dot in the graph corresponds to a subset of t-node attributes preserved in Czech-to-Czech translation via the t-layer. As we remove additional attributes, the achievable BLEU score (the x-axis) decreases, and so does the entropy of predicting the (atomic) label of a t-node regardless any context (the y-axis). A good balance would mean we find a subset of t-node attributes with a high achievable BLEU and a low entropy. Unfortunately, the lower right corner of the graph does not contain any such points.

The graph also indicates some important values of BLEU score and entropy for the given test set. We are surely interested in improving the translation quality beyond what Moses produces.[3] For our particular dataset, Moses achieved BLEU of 12.09 (baseline) to 14.56 (multi-factor setup with additional monolingual data), see the vertical lines in the graph. This makes all dots below

[2]The experiments were carried out on a dataset also used in MT experiments with Moses (Bojar and Hajič, 2008) and using an older version of the analysis/generation pipeline, so the BLEU scores do not match those in Section 4.1.1.

[3]BLEU does not exactly correlate with translation quality as ranked by humans, but it is one of the most widely used metrics.

14.56 BLEU uninteresting from the practical point of view. The horizontal line in the graph represents the entropy of plaintext word forms, taking even letter case information into account. We are not surprised to see that the entropy of (atomic) t-nodes is higher than the entropy of word forms, as t-node attributes capture also information coming from surrounding auxiliary words.

The bad news is the tiny room that was left for improvement: if we assume atomic node labels and we wish to stay below the entropy of plaintext word forms, we cannot obtain BLEU higher than 16.20 due to the inevitable generation step.

To conclude, node labels in the transfer cannot be treated atomically. The attributes of t-nodes have to be considered individually.

4.1.3 Explosion of the Search Space

As described in Deliverable 3.2, we use the approach of factored translation (Koehn and Hoang, 2007) to generate target-side attributes, see Figure 4.4.

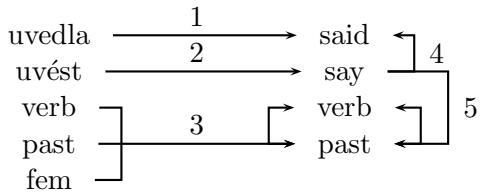

Figure 4.4: Sample decoding steps in word-for-word factored translation.

Our preliminary implementation of factored translation was restricted to treelets containing only one internal node, in the module `t_word_for_word_factored`. We have extended the implementation to `t_structure_factored` to support any number of internal nodes in the treelet (up to the limit global limit of treelet size).

Given a source treelet, the first probabilistic decision made by `t_structure_factored` is to predict the structure of the target treelet: the number of internal nodes, inner edges and also the number and labels of frontier nodes. This decision is conditioned on the input treelet structure and the user can request to consider also the values of some source factors. Subsequent mapping steps convert source factors to target factors in a fixed order following the configuration. In each step, all internal nodes are handled at once: e.g. if the configuration specifies to translate source t-lemma to target t-lemma, and the treelets contain one node in English and two nodes in Czech, both of the target t-lemmas are filled at once (*typewriter* → *psací stroj*).

It is important to note that our implementation of factored translation assumes **synchronous** decomposition of input trees into treelets, much like Koehn and Hoang (2007) assume synchronous segmentation of input sentence into phrases. This means that the input is *first* decomposed to treelets and the whole factored translation of input factors to output factors happens within each of the treelet. Fully specified target treelets are then used in gradual hypothesis expansion.

Given the synchronousness assumption, the whole process of factored translation can be treated as a pre-processing step only, avoiding sparse training data (unseen combinations of attribute values are generated on the fly).[4] Unfortunately, the fact that there are too many target treelet options remains unchanged and leads to a new issue: search errors. While all the various target side attribute values are chosen based on a very limited source context (the corresponding source treelet only), the real utility of individual translation options would become apparent only when the treelet is attached to a partial hypothesis, on the basis of surrounding nodes. Before the attachments happen, many of the translation options had to be pushed off the stack.

[4]The other approach, "asynchronous phrases" in phrase-based MT was not applied with a success yet. With asynchronous phrases (some factors filled and some still pending), it is not quite clear how to implement stack-based pruning and the system falls into many search errors (Philipp Koehn, pers. comm.).

Another consequence of the explosion (and insufficient stack space) is too little variance in the final n-best list. Many hypotheses differ in values of less important node attributes only and the more important choices such as a different t-lemma or formeme been pruned. The surface representation of the hypothesis created by the final generation procedure may not reflect all the subtle differences and in such cases, the stack space was simply wasted.

4.1.4 "Delayed" Factors to Tackle the Combinatorial Explosion

We see two ways of handling the combinatorial explosion of possible output attributes. One of them is to condition target-side treelets not only on the source side treelet, but also on other source-side nodes around the treelet in question. This would steer the scores towards attribute values more relevant for the context, albeit estimated from the source side only.

Another option is to "pack" (or ignore for a while) the ambiguity of less important attributes and choose their values at a later stage, when the overall tree structure has been decided along with key attributes such as t-lemma and formeme. This saves a considerable portion of the stack space (so we see a greater variance in the n-best list) and also allows to condition the values of these attributes on anything from both the input tree and the target hypothesis.

For the time being, we decided to implement the latter option: the module `t_structure_factored` now accepts some output attributes marked as "delayed". Such attributes are filled with a special symbol indicating that the real value has to be determined later, with a pointer to the corresponding source-side node. These translation options are attached together to form hypotheses as usual. Once the main search finishes, an n-best list of hypotheses is extracted from the search graph. All delayed output factors of all hypotheses in the n-best list are filled with the best suggestion as provided by a maximum entropy classifier (we use the implementation by Zhang (2004)) based on user-specified observations of the node's neighbourhood in both the source tree and the current hypothesis.

4.2 Empirical Evaluation

This section describes various configurations of TreeDecode and the performance in terms of BLEU score on WMT09 development set, part b.

4.2.1 Sequence of Back-off Models

The configuration of TreeDecode specifies, which probabilistic models (also called "translation option generators") should be consulted first to find translation options for each possible source treelet. If the given model fails and has nothing to offer, the next model is consulted.

In Deliverable 3.3, Section 3.5.5, we mentioned the option to consult all models and merge their suggestions. A translation option suggested (with some probability) by several models is surely better than a translation option suggested only by one of the model. This means that the scores from translation option generators should default to a the lowest possible value, unless set by a model. Unfortunately, there is no simple way to implement this behaviour correctly, i.e. to score translation options fairly.

The problem is that in practice translation option generators have to heavily prune the space of possible translation options, and they do it even before a translation option is fully constructed. In the end, a translation option suggested by a model would often not get any scores from another model because the other model have pruned it. Some sort of constraint-based search, or "forced generation" would be necessary to ensure that all translation options are scored by all models for a fair comparison. We leave this for further research and stick to the original fixed sequence of back-off models.

Table 4.2 presents the BLEU scores of various back-off models and their sequential combinations. We use the following models:

Back-off Sequence	43k	643k
strall+wfwindep	5.51±0.49	5.26±0.51
strall+strindep+strdelay+wfwall+wfwindep	5.29±0.49	5.09±0.48
strdelay+wfwindep	4.99±0.45	5.30±0.54
strindep+wfwindep	4.86±0.49	5.39±0.59
wfwindep	4.58±0.51	5.09±0.50
strdelay	4.22±0.43	4.39±0.50
strindep	4.09±0.44	4.50±0.55
wfwall	3.81±0.52	4.11±0.50
strall	3.62±0.42	3.81±0.42

Table 4.2: Performance of various sequences of back-off models.

strall uses `t_structure_factored` to predict the structure of the treelet in the first step and to predict all output factors at once in the second step. This model preserves the maximum number of dependencies between individual attribute values but it is prone to data sparseness.

strindep uses `t_structure_factored` to predict the structure of the treelet in the first step and to predict output factors from the corresponding source factors one by one. We do not use any generation steps between output factors, so we do not preserve or promote any dependencies between output attributes. Apart from sparseness caused by treelet structure (structure of internal nodes and number, positions and states of frontier nodes), this model is very general. The drawback of this model is the combinatorial explosion and severe pruning of constructed treelets discussed in Section 4.1.3.

strdelay again predicts the structure using `t_structure_factored` and generates three most important output factors in two mapping steps:

1. `t_lemma` → `t_lemma`, `gram/sempos`
2. `formeme` → `formeme`, `gram/sempos`

The fact that both the step predict the value of the semantic part of speech (noun, verb, ...) ensures that the formeme types match the declination or conjugation capacities of the `t_lemma`. All other output factors are "delayed" (Section 4.1.4) and their values are determined only once the full hypotheses are constructed based on all factors of the corresponding source nodes.

wfwall uses `t_word_for_word_factored` (see Deliverable 3.3, p.30) to translate treelets of one internal only, preserving the number and order of frontier nodes. The attributes of the output internal node in this "wfwall" configuration are constructed in a single step, preserving all dependencies between their values at the risk of sparse data.

wfwindep also uses `t_word_for_word_factored`, so one source node is translated to one target node and the order of frontier nodes is preserved as well. The target attributes are constructed step by step. As in "strindep", we do not enforce any dependencies as we are not using any generation steps.

Table 4.2 confirms our expectations on two datasets: 43k and 643k sentences. The data sparseness problem strikes hard if "strall" is used alone: in this setting TreeDecode has to see the exact treelet structure and the exact combination of attribute values in the training data or back-off to the very default of simply copying the source node.

The model "wfwall" performs better, confirming that the t-layer is a reasonable abstraction in terms of number of nodes and tree structure: when translating from English to Czech, it is

not that harmful to preserve the whole tree structure. The same observation is confirmed once more by the fact that "wfwindep" performs better than "strindep" and "strdelay".

The delayed vs. independent approach to generation of detailed attribute values does not seem to have a reliable effect. For different datasets, different settings work better. We attribute this to relatively small stack limits so independent generation in the end offers usually the same single-best values as delayed generation.

Not surprisingly, the best combination is to try a detailed model first (e.g. "strall") followed by a reasonable back-off to avoid the trap of sparse training data. On the 43k dataset, the best combination is "strall+wfwindep", on the 643k dataset, "strindep+wfindep" work better.

Note that the little or no difference between the 43k and the 643k dataset is discussed in Section 4.2.4 below.

4.2.2 Rescoring Final Hypotheses based on LM

The final generation of the Czech sentence from the estimated tectogrammatical tree is a deterministic procedure (Žabokrtský et al., 2008) with no access to an n-gram language model. Being aware of the importance of language models for manual and especially automatic evaluation, we decided to implement n-gram-based rescoring of final hypotheses.

If specified in the configuration, TreeDecode now extracts the full n-best list of tree hypotheses, passes all of them to TectoMT for generation of their surface forms, tokenizes the outputs and rescores them using the standard n-gram language model. We use IrstLM (Federico and Cettolo, 2007) for LM estimation and scoring of hypotheses.

	No Rescoring	**LM Rescoring**
43k, no "`#PersPron;`"	4.53±0.40	4.71±0.43
600k+43k, no "`#PersPron;`"	4.37±0.45	4.55±0.44
600k, no "`#PersPron;`"	3.82±0.41	3.80±0.39
84k	4.49±0.40	4.69±0.43

Table 4.3: LM Rescoring helps, except for the 600k out-of-domain training dataset.

See Table 4.3 for an evaluation of the impact of n-gram language model rescoring on "etct" translation trained on varying data sizes. In all experiments, we used the same LM based on the 84k sentences. Albeit within the confidence intervals, LM rescoring proves to improve the BLEU scores, with the exception of out-of-domain parallel training data, see Section 4.2.4 for more details.

4.2.3 Alignment of T-Nodes for Heuristic Treelet Extraction

Tree Aligner (Section 3) implements a method to decompose pairs of trees into aligned treelets (STSG rules). The treelet alignment method in principle does not need any alignment of nodes in the trees, although such a node alignment can be used to prune unlikely treelet alignments.

Due to the poor coverage of treelet translation rules extracted by Tree Aligner, we resort to the treelet extraction heuristics described in Deliverable 3.2 (Section 3.4): we extract all treelet pairs compatible with node-to-node alignment up to a maximum size of a treelet.

In the experiments reported in Deliverable 3.2, we had to use node alignments obtained by applying GIZA++ (Och and Ney, 2000) to linearized tectogrammatical trees.[5] Since then, a new method for alignment of t-nodes was developed by Mareček et al. (2008).

Table 4.4 confirms that the improved alignment increases also the quality of our "etct" transfer in various settings, albeit still within the empirical confidence intervals.

[5] We run GIZA++ on sequences of t-lemmas in linear order as appearing the trees, which need not correspond to the surface word order. We symmetrize the two GIZA++ runs using intersection as we prefer precision to recall in alignments.

Training data size and options	T-Alignment	GIZA++ Intersection
43k, no "`#PersPron;`", big rescoring LM	4.92±0.44	4.79±0.46
43k, no "`#PersPron;`"	4.71±0.43	4.69±0.44
84k	4.69±0.43	4.54±0.43

Table 4.4: T-Alignment performs better than GIZA++.

We noticed that the current implementation of the t-alignment performs extremely poorly on nodes with no real t-lemma such as nodes representing pronouns: "`#PersPron;`". A manual evaluation of a tiny sample revealed that 9 out of 11 "`#PersPron;`" nodes were mis-aligned and the remaining two were correctly not aligned to anything.

Based on this observation, we restricted our baseline dataset of about 84k sentences to sentences with no "`#PersPron;`". Obviously, personal pronouns appear in many sentences, so the overall data size shrinks to a half: 43k. Still, the overall performance of the setup is higher if sentences with mis-aligned pronouns are ignored.

4.2.4 Harder Domain Dependence

Table 4.5 documents a rather counter-intuitive result that using much larger data (600k of parallel sentence; sentences with personal pronouns not included) can lead to worse translation quality. The difference is more pronounced for transfer at the t-layer than for phrase-based translation.

	etct	epcp
43k, no "`#PersPron;`"	4.71±0.43	7.29±0.66
84k	4.69±0.43	8.32±0.65
600k+43k, no "`#PersPron;`"	4.55±0.44	8.30±0.76
600k, no "`#PersPron;`"	3.80±0.39	7.29±0.78

Table 4.5: Larger data may decrease the performance, and more so for the tectogrammatical transfer.

We already mentioned that the loss in performance from the 43k to 84k sentences is caused by mis-aligned personal pronouns at the t-layer (and we do not see a loss for phrase-based "epcp").

However, the further loss when going to 600k of training data can be attributed only the difference of domains (legal texts, not articles), with the distribution of t-node types and attributes skewed so much, that even an unweighted combination of the 600k and 43k training datasets does not fix the issue.

The huge domain difference is confirmed also by "epcp": the 600k sentences, when used alone, lead to the same performance as the small 43k corpus, but we see a clearly positive effect of plain unweighted combination of the corpora: 600k+43k performs better than 43k.

4.3 Comparison with Moses and TectoMT

Table 4.6 presents the results of our participation at WMT09 (Bojar et al., 2009) complemented by the tree-transfer using TreeDecode. The scores of all systems are based on the official WMT09 test set of 3027 sentences and evaluated by the NIST scoring tool[6]. Note that only the best system of each contributor was manually judged. Our best performing system was a configuration of Moses, "Moses T".

[6] `ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v11b.pl`

For a full description of our configuration of Moses as well as TectoMT, please see Bojar et al. (2009). Here we restrict ourselves to a summary of differences from our tree-based transfer.

TectoMT (Žabokrtský et al., 2008; Bojar et al., 2009) is a highly modular hybrid MT system with transfer at the tectogrammatical layer. Our tree-based transfer shares the pre- and post-processing pipelines with TectoMT but differs in the transfer step. While we aim at learning all the tree-to-tree correspondences automatically for a more or less uniform model from parallel treebanks, TectoMT is a complex hand-coded sequence of transfer rules. Some of these rules rely on probabilistic dictionaries extracted from same the data sources as we use, but most of them are heuristics tailored to the properties of the t-layer.

Moses (Koehn et al., 2007) is a state-of-the-art phrase-based decoder capable of explicitly modelling some linguistic features using additional "factors" in input or output streams of tokens. We use various configurations of Moses and rely on large training data to a great extent. Phrase-based translation can be approximated with TreeDecode using linear trees (right branching only) of word forms, denoted as "epcp". The main limitation of this approximation is no possibility of phrase reordering. All word reordering has to happen within phrases. For the sake of comparison, we report also vanilla phrase-based translation performed by Moses trained on the same dataset, using the same language model and with only distance-based reordering model. We see that with no model optimization (MERT), our "epcp" and Moses deliver translations of nearly identical quality.

System	**BLEU**	**NIST**	**Rank**
Moses T	**14.24**	**5.175**	-3.02 (4)
Moses T+C	13.86	5.110	–
Google	13.59	4.964	-2.82 (3)
U. of Edinburgh	13.55	5.039	-3.24 (5)
Moses T+C+C&T+T+G 84k	10.01	4.360	–
Eurotran XP	09.51	4.381	-2.81 (2)
PC Translator	09.42	4.335	**-2.77 (1)**
TectoMT	07.29	4.173	-3.35 (6)
TreeDecode **epcp** 84k	08.07	3.942	–
TreeDecode **etct** 643k	05.53	3.660	–
TreeDecode **etct** 43k	05.14	3.538	–
Vanilla Moses 84k, even weights	08.01	3.911	–
Vanilla Moses 84k, MERT	10.52	4.506	–

Table 4.6: Automatic scores and preliminary human rank for English→Czech translation. Systems in italics were developed by other teams and are provided for comparison purposes only. Best results in bold.

Unfortunately, the results in Table 4.6 suggest that simpler models perform better, partly because it is easier to tune them properly both from computational point of view (e.g. MERT not stable and prone to overfitting with more features[7]), as well as from software engineering point of view (debugging of complex pipelines of tools is demanding). Consider also the variance in BLEU scores due to different model configurations in Section 4.2.1: an extensive "tuning" of a tree-to-tree transfer system would examine not only the few basic sequences of back-off models but also various decoding sequences in all factored models and the impact of all implemented stack limits. We imagine this process could be automated in a fashion similar to MERT, except that instead of optimizing in a contiguous vector space of weights, we would be optimizing in a hierarchical discrete space of configurations. However, we have to leave this idea for further research.

[7]For "Moses T+C+C&T+T+G", we observed BLEU scores on the test set varying by up to five points absolute for various weight settings yielding nearly identical development set scores. For tree-based transfer, MERT does not converge at all and hardly ever suggests any combination superior to the default even weights.

Apart from an easier tuning, we observe that simpler models run faster: "Moses T" produces 12 sentences per minute, which is 4.6 times faster than "Moses T+C". (Note that we have not tuned either of the models for speed.) The "etct 643k" needs about 49 seconds for one sentence of which 46 are spent in generating surface representations of n best hypotheses.

While "Moses T" is probably nearly identical setup as Google and Univ. of Edinburgh use, the knowledge of correct language-dependent tokenization and the use of relatively high quality large language model data seems to bring moderate improvements.

It is also worth mentioning that while our "Moses T" seemed to beat Google and all other systems in terms of BLEU and NIST, we actually ranked slightly worse in manual evaluation. The most striking difference between BLEU and human Rank is for two Czech commercial MT systems, PC Translator and Eurotran XP.

Because none of our TreeDecode systems was manually ranked at WMT09 and because BLEU has proved to correlate poorly with human judgements, we performed a quick manual comparison of "Moses T" and "etct 643k". For this particular pair of systems, we have to confirm BLEU scores: "Moses T" delivered a reasonable (for MT) translation that was clearly more fluent than "etct" in 69% of sentences while "etct 643k" produced a barely acceptable translation (and nearly always worse than Moses) in only 38% of cases.

Chapter 5

Tectogrammatical Layer for MT Quality Evaluation

In an independent line of research, we employed the tectogrammatical layer in a tool that automatically estimates MT quality. The full report is available in Kos and Bojar (2009).

The newly introduced metric is called **Semantic POS Overlapping** (SemPOS) and it is inspired by Giménez and Márquez (2007) who employ various linguistic features from the syntactic or semantic representation of the sentence. One of their best performing metrics was *semantic role overlapping*. Since we did not find a tool that would assign semantic roles to words in a Czech sentence, we decided to slightly modify the idea. The TectoMT framework (Žabokrtský et al., 2008) can assign a semantic part of speech (semantic POS) to words. We compute the **overlapping** (as defined in Giménez and Márquez (2007)) of the hypothesis with the reference for this linguistic feature. Moreover, we do not use the surface representation of the words but their **t-lemma** obtained from the TectoMT framework. As an approximation, we can say that SemPOS evaluates the lexical choice of autosemantic words, taking the (semantic) part of speech into account so ambiguous words are not confused (*to state* vs. *the state*). The fluency of the hypothesis is checked only indirectly, by the TectoMT pipeline analyzing the hypothesis.

Metric	**Articles**	**Editorials**	**Average**
NIST	0.22±0.60 (7)	0.26±0.62 (1)	0.24
F-measure/GTM(e=1)	0.24±0.58 (1)	0.23±0.63 (4)	0.23
GTM(e=0.5)	0.24±0.58 (2)	0.23±0.63 (6)	0.23
GTM(e=2)	0.24±0.58 (3)	0.22±0.63 (10)	0.23
Meteor	0.23±0.57 (4)	0.24±0.62 (2)	0.23
GTM(e=0.1)	0.23±0.58 (5)	0.23±0.63 (5)	0.23
Meteor(orig)	0.23±0.57 (6)	0.23±0.62 (7)	0.23
PER	0.22±0.60 (8)	0.24±0.63 (3)	0.23
TER	0.21±0.60 (9)	0.23±0.62 (8)	0.22
WER	0.21±0.60 (10)	0.23±0.62 (9)	0.22
SemPOS	0.21±0.57 (11)	0.19±0.61 (11)	0.20
BLEU	0.03±0.63 (12)	0.02±0.62 (12)	0.03

Numbers in brackets indicate the relative position of the metric.

Table 5.1: Average sentence-level correlations for the metrics including standard deviation.

Tables 5.1 and 5.2 summarize all the MT quality metrics evaluated by Kos and Bojar (2009) for English→Czech MT in terms of Pearson correlation coefficient against human judgments. The experiments were conducted with two data sets (Articles and Editorials) on sentence-level and system-level scale. (For system-level comparison there are no ties and Pearson correlation coefficient equals Spearman's rank correlation coefficient.)

Metric	Articles	Editorials	Average
SemPOS	**0.81±0.18 (1)**	**0.75±0.23 (1)**	0.78
Meteor	0.43±0.18 (2)	**0.60±0.28 (2)**	0.52
Meteor(orig)	0.43±0.18 (3)	**0.52±0.32 (3)**	0.47
GTM(e=0.1)	0.24±0.34 (9)	*0.48±0.34 (4)*	0.36
GTM(e=0.5)	0.40±0.22 (5)	0.28±0.33 (5)	0.34
BLEU	0.40±0.23 (6)	0.25±0.33 (6)	0.33
F-measure/GTM(e=1)	0.41±0.21 (4)	0.21±0.31 (7)	0.31
GTM(e=2)	0.31±0.34 (7)	0.18±0.31 (9)	0.24
NIST	0.25±0.34 (8)	0.21±0.31 (8)	0.23
PER	0.01±0.38 (10)	0.16±0.32 (12)	0.09
TER	-0.17±0.41 (11)	0.18±0.32 (10)	0.00
WER	-0.17±0.41 (12)	0.18±0.32 (11)	0.00

Results covered in the error bounds of the best result are in bold. Results covering the best result in their error bounds are in italics. Numbers in brackets indicate the relative position of the metric.

Table 5.2: Average system-level correlations with standard deviations for the metrics computed from bootstrapped samples (N=10000).

Our results confirm the well known fact that BLEU correlates poorly with human ranking at the sentence level and achieves moderate agreement on the system level. Better-performing metrics for system-level comparison include Meteor (Banerjee and Lavie, 2005) and GTM (Turian et al., 2003) but SemPOS appears to be the clear winner. A similar observation was confirmed on a third dataset at WMT09 (Callison-Burch et al., 2009).

We plan to extend this research and explicitly score also the grammatical coherence of MT output, albeit heavily influenced by the choice (and errors) of the parser.

Chapter 6

Conclusion

In this deliverable, we described further improvements of our tree-based aligner and transfer system aimed at machine translation via the tectogrammatical layer.

We documented that the hard constraint of parallel syntactic structures of source and target sentences causes serious problems when the model is applied to real data, both manually and automatically analyzed: only 10 to 25 % of sentences can be reconstructed using the synchronous tree-substitution grammar as extracted by our tree aligner.

An empirical analysis of the transfer at the deep syntactic representation of the sentence revealed the problem of detailed node attributes required in order to generate a reasonable surface form of the sentence. We have further extended our tree-based transfer to allow simultaneous changes of tree structure as well as the generation of detailed output attributes (previous versions could focus on structure or output attributes but not both features of output treelets at the same time). We also implemented the ability to rescore final linearized hypotheses, so that a traditional n-gram language model can be applied to the output of the deterministic generation from tectogrammatical trees.

We presented and evaluated several configurations of the tree-based transfer with significant improvements in BLEU score. Unfortunately, a contrastive comparison with our configuration of a phrase-based model reveals that the tree-based approach produces hypotheses of much worse quality. The reason lies in the complexity of sentence structure, made apparent by the formal representation at the tectogrammatical layer. Despite our efforts, we were not able to pinpoint the correct independence assumptions that would strike a balance between modelled level of detail and data sparseness.

While our results so far confirm the success of rather simple phrase-based models, we do believe in the potential of our deep syntactic representation of the sentence.

One of the side achievements of this project is the application of a pipeline of automatic tools on huge datasets. We will continue experimenting with exploitation of the large-scale rich annotation both for phrase-based as well as for syntax-based approaches to MT. The data was already partially used in our experiments and served as valuable resource for an independent team of colleagues developing TectoMT, a more heuristical approach to MT at the tectogrammatical layer.

Another important result is the application of the tectogrammatical layer in evaluation of MT quality when translating to Czech (Section 5). An automatic metric based on features from the t-layer scored best in terms of correlation with human judgments on several official datasets. We will continue this direction of research and experiment e.g. with the correlation then evaluating translations to English and also with adding further features from the t-layer such as preserved dependencies.

By implementing the tree aligner and the tree transfer system and applying them to real data, we learned a lot about statistical search in complex structured spaces. Our plan for the near future is to investigate methods capable of combining shallow and deep features in a unified search. The most important property of such a system is to seamlessly back-off to simpler models.

Appendix A

References

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization at the 43th Annual Meeting of the Association of Computational Linguistics (ACL-2005)*, pages 65–72, Ann Arbor, Michigan, June.

Ondřej Bojar and Martin Čmejrek. 2007. Mathematical Model of Tree Transformations. Project Euromatrix - Deliverable 3.2, ÚFAL, Charles University, December.

Ondřej Bojar and Jan Hajič. 2008. Phrase-Based and Deep Syntactic English-to-Czech Statistical Machine Translation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 143–146, Columbus, Ohio, June. Association for Computational Linguistics.

Ondřej Bojar, Miroslav Janíček, Zdeněk Žabokrtský, Pavel Češka, and Peter Beňa. 2008a. CzEng 0.7: Parallel Corpus with Community-Supplied Translations. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May. ELRA.

Ondřej Bojar, Miroslav Janíček, and Miroslav Týnovský. 2008b. Implementation of Tree Transfer System. Project Euromatrix - Deliverable 3.3, ÚFAL, Charles University.

Ondřej Bojar, David Mareček, Václav Novák, Martin Popel, Jan Ptáček, Jan Rouš, and Zdeněk Žabokrtský. 2009. English-Czech MT in 2008. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, Athens, Greece, March. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder. 2009. Findings of the 2009 workshop on statistical machine translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, Athens, Greece. Association for Computational Linguistics.

Martin Čmejrek, Jan Cuřín, Jiří Havelka, Jan Hajič, and Vladislav Kuboň. 2004. Prague Czech-English Dependecy Treebank: Syntactically Annotated Resources for Machine Translation. In *Proceedings of LREC 2004*, Lisbon, May 26–28.

Marcello Federico and Mauro Cettolo. 2007. Efficient Handling of N-gram Language Models for Statistical Machine Translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 88–95, Prague, Czech Republic, June. Association for Computational Linguistics.

Jesús Giménez and Lluís Márquez. 2007. Linguistic Features for Automatic Evaluation of Heterogenous MT Systems. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 256–264, Prague, June. Association for Computational Linguistics.

Jan Hajič. 2004. Complex Corpus Annotation: The Prague Dependency Treebank. In *Insight into Slovak and Czech Corpus Linguistics*, Bratislava, Slovakia. Jazykovedný ústav Ľ. Štúra, SAV.

Philipp Koehn and Hieu Hoang. 2007. Factored Translation Models. In *Proc. of EMNLP*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej

Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of EMNLP 2004*, Barcelona, Spain.

Kamil Kos and Ondřej Bojar. 2009. Evaluation of Machine Translation Metrics for Czech as the Target Language. *Prague Bulletin of Mathematical Linguistics*, 92. in print.

David Mareček, Zdeněk Žabokrtský, and Václav Novák. 2008. Automatic Alignment of Czech and English Deep Syntactic Dependency Trees. In *Proceedings of EAMT 2008*, Hamburg, Germany.

Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolářová, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Ševčíková, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, and Zdeněk Žabokrtský. 2007. Annotation on the tectogrammatical level in the Prague Dependency Treebank. Project Euromatrix - Deliverable 3.1, ÚFAL, Charles University, May.

Franz Josef Och and Hermann Ney. 2000. A Comparison of Alignment Models for Statistical Machine Translation. In *Proceedings of the 17th conference on Computational linguistics*, pages 1086–1090. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *ACL 2002, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania.

Joseph P. Turian, Luke Shen, and I. Dan Melamed. 2003. Evaluation of Machine Translation and its Evaluation. In *Machine Translation Summit IX*, pages 386–393. International Association for Machine Translation, September.

Zdeněk Žabokrtský and Ondřej Bojar. 2008. TectoMT, Developer's Guide. Technical Report TR-2008-39, Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague.

Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. 2008. TectoMT: Highly Modular Hybrid MT System
with Tectogrammatics Used as Transfer Layer. In *Proc. of the ACL Workshop on Statistical Machine Translation*, pages 167–170, Columbus, Ohio, USA.

Le Zhang. 2004. Maximum Entropy Modeling Toolkit for Python and C++. `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`.