# WiDec Description Paper

Tsimafei Prakapenka, March 2022.

## Problem Description

### Introduction

Prior to huge popularity of Neural Machine Translation, statistical approaches were considered as state-of-the art models for this task. The goal of this project is to implement one of the steps for Statistical Machine Translation (SMT) - **decoding**. Particularly, this project is devoted to **phrase reordering**. This problem is interesting, since the task, in general, is a NP-complete search problem. To simplify things a bit, It means that complexity of the naive algorithm is exponential, so for long sentences it will be running for a very long time. This project implements two decoding algorithms, which provide a reasonable balance between search accuracy and speed.

This project was inspired by HW3 (description, github) from JHU Machine Translation class. Authors provide:

- set of 48 French sentences to test the system on
- language model and translation model
- baseline system capable of monotone decoding without reordering
- evaluation script to measure quality of produced translations

All base components will be described and explained later in this paper.

Speaking about technical aims of this project, following algorithms were implemented:

- ***Beam Search*** capable of reordering with the additional following features:
  - stack pruning (both histogram and threshold pruning available)
  - reordering limits

- ***Greedy Hill Climbing***
  - naive word-to-word, monotone and beam decoding seeds
  - swap, replace, bi-replace, merge and split operations

Additionally, I reorganised code provided by authors of HW3 and added comments.

# SMT Theory Review

Before I will describe an actual SMT system, let us describe major theoretical components of the system.

Except of reordering, other steps of SMT process include language modelling (LM - language model) and phrase-to-phrase translation (TM - translation model). LM evaluates how plausible is the given sentence in terms of the selected language. TM measures probability of the translation of one specific phrase in source language to the specific phrase in target language. Both these models can be trained using alignment algorithms (MGIZA, fast_align, the Berkeley aligner).

So, being aware of LM and TM, important components of every SMT model, we will introduce other important concepts. Please note, that I am heavily relying on slides by Philipp Koehn.

1. Basic Translation Idea.

We need to align all source phrases to some target phrases (Figure 1). Please note, that multiple source words can be aligned to one target word and vice versa. Also, it is important that source phrases can be reordered in target language to follow the rules of other grammar.

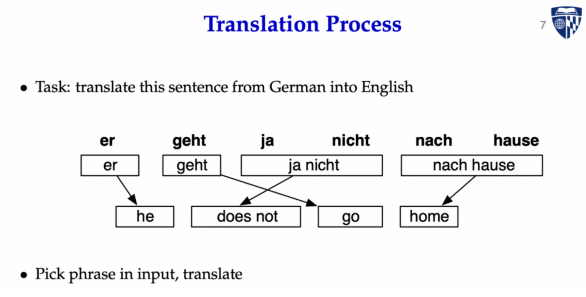


Figure 1. Translation process.

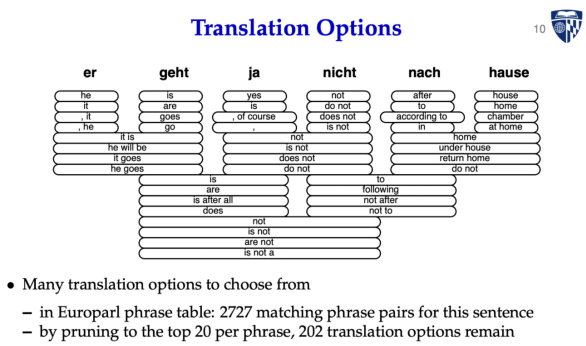


Figure 2. Translation Options.

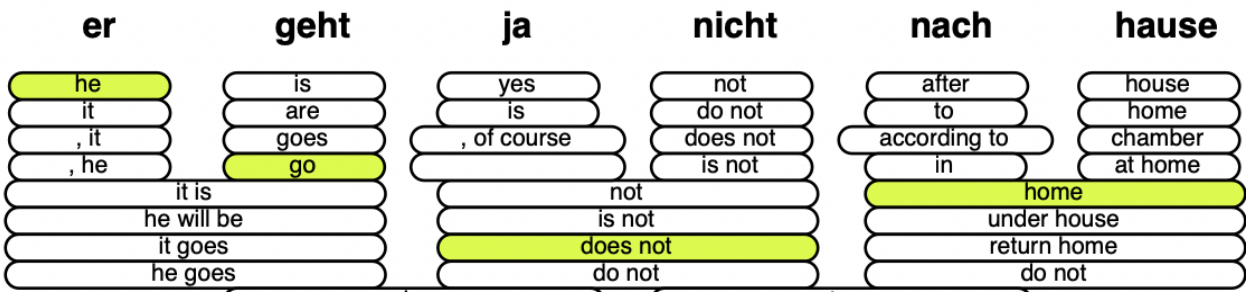


Figure 3. Picking Right Phrases.

2. Translation Table.

   It is an output of Translation Model. Basically, it provides probabilities of actual translations for different phrase coverage of the source sentence. The problem is that search space is very huge considering all possible options. However, oftentimes 10 or 20 translations per phrase are enough. Allowing more options radically increases time of running and not improves the quality much. Hence, pruning is applied, as described on Figure 2.

3. Beam Search

Having phrase translation table, we need to solve two problems:
– picking the right translation options (Figure 3, not that they are not top-1 for each phrase)

– arranging them in the right order (without reordering we will end up with the sentence ‘he go does not home’, which is grammatically incorrect in English)

Thus, it is a search problem that can be solved using heuristic beam search:

1. Consult phrase translation table for all input phrases (taking pruning into account).
2. Initial hypothesis: no input words covered, no output produced.
3. Pick any translation option from table, create new hypothesis. Create other hypotheses starting

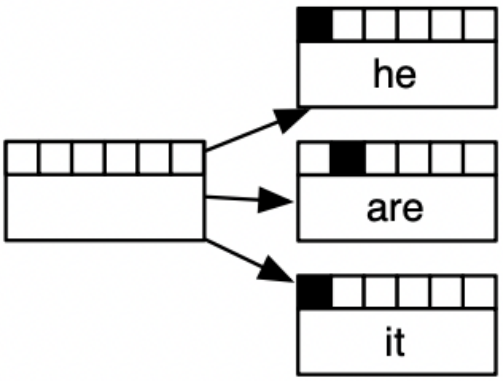


Figure 4. Hypotheses Expansion.

from initial one. Bit mask is used to show which words from source have been translated. The search terminates when all source words are translated.

4. Create hypotheses from created partial hypothesis, then backtrack from the best option (Figure 5).

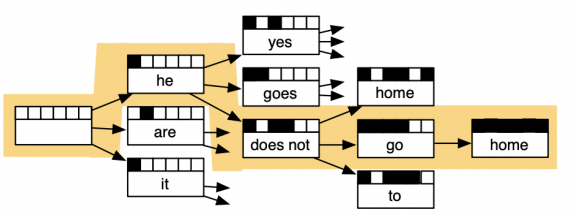


Figure 5. Best Hypothesis Backtrack Path.

It is very important to understand that we starting from a random translation at random position. For example, we can translate punctuation mark from end of the sentence first or translate a selected phrase/word with another option from the translation table. For example, taking ‘he’ as a translation for German word ‘er’ produces new hypothesis comparing to taking ‘it’ as a translation. So, search space is exponentially big and some kind of optimisation is required.

4. Stacks.

One of the pruning techniques includes stacks. The idea is the following:

- put comparable hypothesis into stacks (same words translated - Figure 6)
- limit number of hypotheses in each stack

Limitation is done either by histogram pruning (keep at most k hypotheses in each stack) or threshold pruning (keep hypothesis with score α × best score (α < 1)). In our algorithms we use mostly histogram pruning, however threshold pruning is also implemented.

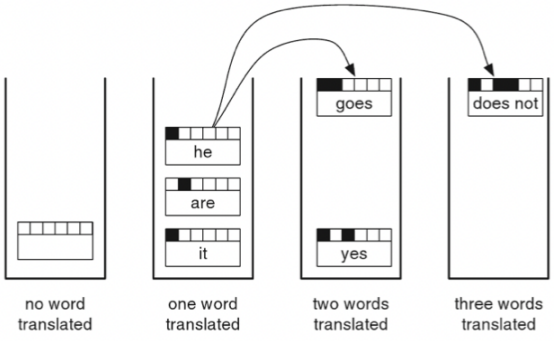


Figure 6. Stack Decoder.

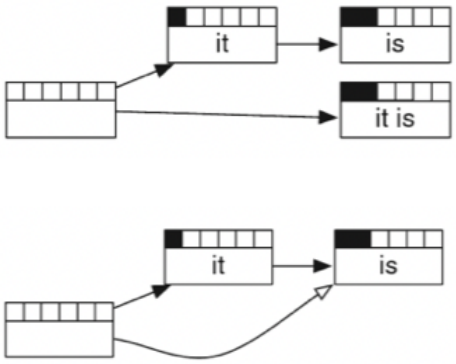


Figure 7. Recombination.

5. Recombination

Two hypothesis paths lead to two matching hypotheses: they have the same number of foreign words translated and the same English words in the output, but different scores. Thus, in the heuristic search we drop the worse hypothesis (Figure 7). It is a risk-free version of pruning.

Now we are equipped with all theory, that we need at the moment. So, let us look at actual implementation of decoding systems.

# System Description

## Given System

The baseline decoder, which is not capable of reordering, is provided in HW3 from mt-class.org.

There are two python programs here:

- `decode` translates input sentences from French to English.

It is a basic algorithm without phrase reordering. Produces a sample translation to stdout. Algorithm will be described in details later.

- `compute-model-score` computes the model score of a translated sentence.

These commands work either in a pipeline or separately using helper files. For example:

```
> python decode | python compute-model-score
# auxillary file can help us to have a look at produced translations
> python decode > translations.txt
> python compute-model-score < translations.txt
```

There are also some helper programs and files:

- `models.py` implements very simple interfaces for language models and translation models.

The `data` directory contains files derived from the Canadian Hansards:

- `input`: 48 French sentences to translate.

- `tm`: a phrase-based translation model. Each line is in the form:

  French phrase ||| English phrase ||| log_10(translation_prob)

- `lm`: a trigram language model file in ARPA format (more details about the format here).

  log_10(ngram_prob) ngram log_10(backoff_prob)

The language model and translation model are computed from the data in the align directory, using alignments from the Berkeley aligner.

Our implementations ('widecode' and 'widecode_greedy') were designed in the same way as default 'decode' script, so you can use them in pipeline with scoring script. For specific values of parameters please run scripts with -h flag ('python3 widecode -h').

## Default Decoding Algorithm

Now, we want to introduce pseudocode for default decoding algorithm:

```
Input: f_sentence - sentence to translate;

1 n = num_words_in_f_sentence
2 create stacks collection of size of (n + 1)
3 place empty hypothesis into stack 0
4 for each stack_position i from 0 to n − 1:
5   for each hypothesis in stacks[i]:
6     for each end_of_span j from i+1 to n:
7       # take translation options from TM
8       for each translation_option for f_sentence[i:j]:
9         create new hypothesis
10        place in stack
11        recombine with existing hypothesis if possible
12        prune stack if too big

13 select best translation by the most probable hypothesis from the last stack
14 backtrack to reconstruct english sentence from the best hypothesis
```

Notes:

- The most important thing here is that i-th stack always covers first-i words. It can be completed with different phrase splitting, but we will never skip words. It is a major difference with a beam search model, capable of reordering.
- Stack pruning can also be conducted on step of selecting hypotheses by selecting top-s most probable hypotheses.

- Hypothesis object has the following fields:
  - `logprob` - probability of translation up to the current hypothesis in log scale.
  - `lm_state` - state of LM, which helps to evaluate probability in terms of target language after adding new phrase translation to previous hypothesis. In our system, this evaluation is done based on trigram model.
  - `predecessor` - previous hypothesis, allowing later to backtrack from the best translation
  - `phrase` - English translation phrase produced by hypothesis
  - `fi` - left index (inclusive) of source French phrase translated by hypothesis
  - `fj` - right index (exclusive) of source French phrase translated by hypothesis

## Beam Search Decoding Algorithm

Beam Search decoding algorithm is based on monotone decoding with several important additions:

- stack thresholding by alpha parameter
- reordering penalty (monotone decoding has 0 reordering penalty)
- reordering limits (restrict reordering longer than n words, usually 5-8)

```
lines 1-5 from monotone decoding
for span_width from 1 to n - i:
  for span_start fi from 0 to n-1:
    span_end = min(fi + span_width, n + 1)
    if f_sentence[fi:fj] is applicable to h:
      check_reordering limits
      add reordering penalty
lines 7-13 from monotone decoding
```

Some comments for the algorithm above:

- The main difference with default algorithm is an ability to consider all possible translations from every stack, not only monotonous ones.
- if applicable means that new translation option does not cover already translated words. In actual implementation we reach it by introducing a bitmask of covered words. This bitmask is added as a field to hypothesis object.

To finalise system review part, we would like to tell a few words about scoring script, which evaluates produced translations. It takes two files of the same length as an input: French source and English target (can also be passed via standard input, like UNIX pipeline).

Later, there are two mains steps:

1. Produce alignments
    a. Generate all possible french phrases (spans) from source sentence.
    b. Iterate by all possible English translations for each French phrase (here is no TM limit) and check whether translations are present in target phrase.
    c. Store all french-english alignments that exist in target phrase
2. Compute sum of probability of all possible alignments by dynamic programming

    To do this, we recursively compute the sum over all possible alignments for each pair of English prefix (ei) and French mask (v), working upwards from the base case (ei=0, v=0) [i.e. forward chaining]. The final sum is the one obtained for the pair (ei=len(e), v=range(len(f)).

This is only brief description of the scoring algorithm, since it is given for granted by mt-class sources. I just want to make theory clearer. Also, I have added extensive comments in code, so you are welcome to check it.

## Greedy Hill Climbing

Greedy Hill Climbing is a completely different approach to solving decoding problem. It is described in detail by Philipp Koehn and Langlais et.al (2007).

First, I would like to provide pseudocode and then describe the algorithm itself.

```
Input: source - sentence to translate

current = seed(source)

loop:
  current_score = score(current_translation)
  best_score = current_score
  best = current_translation
  for all h in neighbours(current) do
      candidate_score = score(h)
      if candidate_score > best_score:
```

```
            best_score = candidate_score
            best = h
    if current_score == best_score:
        return current
    else:
        current = best
```

1. We generate rough initial translation by seed function. It can is as simple as word-by-word translation without reordering. Alternatively, we can use any other decoding algorithm like monotone decoding.
2. Score seed translation somehow using LM and TM. We use a function from compute-model-score script.
3. Apply a number of changes to improve it - neighbours function in pseudocode. Steps are described in detail by Langlais et.al (2007).
    a. Move: move English phrase to the other place in sentence
    b. Swap: swap two adjacent target phrases
    c. Replace: change the translation of a phrase to any other from TM
    d. Bi-Replace: change the translation of two adjacent phrases
    e. Split: split up the translation of a phrase into two smaller translations
    f. Merge: Combine two adjacent source phrases to the single phrase and translate it

We apply the above-mentioned steps iteratively until no improvement can be achieved.

Main advantages of this approach are:

- we always have a full translation, so we can terminate search every moment and have some kind of resulting translation (e.g. maximum 5 seconds of decoding per sentence)
- we can also score global properties of sentence, like presence of verb, since we have full sentence every moment.

Main disadvantage is a small search space, since we can stuck in local optima, requiring two or more steps to solve. Beam search avoids this kinds of problem by keeping even less probable hypothesis than current, so we can use them later to find a global maximum.

# Evaluation

We conducted series of experiments, summarised in the table above. We also store produced files in ‘model_translations’ directory, so you can reproduce evaluation scores or conduct human assessment of quality.

| | 1: (1, 1) | 2: (10000, 1) | 3: (10000, 10) | 3: (10000, 20) | 4: (10000, 200) |
|---|---|---|---|---|---|
| decode | (0.007; -1439.9) | (0.007; -1436.3) | (0.67;-1354.64) | (2.77; -1353.67) | (18.68; -1353.24) |
| widec-py | (0.41; -1749.9) | (55.9; -1459.01) | (1470.6; -1333.8) | (2914.66; -1341.06) | (4385.7; -1339.1) |
| widec-py-reord | | | (1556.54;-1351.21) | (3567.5; -1344.1) | |
| widec-greedy | | | (2135.79;-1323.8)<br>(5107.08;**-1298.2**) | | |

Row name states the name of the decoder algorithm:

- decode - monotone decoding algorithm, provided by the authors of HW3.
- widec-py - beam search algorithm with stack size pruning, so no stack threshold pruning, reordering limits or additional reordering penalty added.
- widec-py-reord - modified version of widec-py with distance-based reordering penalty added to the TM and LM scores. Base value for all experiments is 0.75.
- widec-greedy - implementation of greedy hill climbing algorithm.

Column names format is the following: (max_stack_size, max_translations_per_phrase)

In cells we store decoding time and resulting score. For Greedy Hill Climbing we report both monotone seed (first result) and beam heuristic search seed (second result). Evaluation time varies from 35 seconds to 106. However, comparing to decoding time it is insignificant, so we do not report it. Also, this table has empty cells, since we were interested in comparing our algorithms with best search hyperparams instead of time-consuming full scan.

We always evaluated produced translations of full corpus of French sentences (48 entries) by compute-model-score script. The resulting scores are in log-scale, so lower values are better.

Conclusions:

1. Algorithms, which are capable of reordering, work better than monotone decoding algorithms. However, they work much longer, and with serious restrictions of stack size and translations per phrase perform much worse than default algorithm (-1749.9 vs. -1439.9).
2. In general, increasing stack size and number of translations per phrase improves quality of every algorithm. Nevertheless, we got the best score for ‘widec-py’ with 10 translations per phrase, 20 and 200 translations show worse quality. Probably, it is a data-specific issue.
3. Surprisingly, ‘widec-py-reord’ performs worse that the basic ‘widec-py’. We expected that penaltising reordering will improve the quality of the model.
4. Greedy Hill Climbing is able to significantly improve monotone decoding (-1354.64 vs. -1323.8) and even outperform beam search having comparable time of run. This algorithm gave us the best result -1298.2 using beam seed. I have also tried to run this algorithm with word-by-word translation as a seed. Running time is 7350 seconds and quality is -1650.
5. It looks like that there is no need to consider more than 10 translations per phrase. With value of 20 and 200 we have a subtle increase in performance, but serious drop in running time.

I have also experimented with compiling Python code to binary file, but execution time is roughly the same as running the program as a script, so we do not report it.

# Future Work

I see the following directions of future work:

1. Do careful grid search of optimal combination of hyperparameters: maximum stack size, stack threshold, maximum number of translations per phrase, reordering penalty base.
2. Implement C++ version of beam search algorithm to reduce search time.
3. Add more data to test to avoid dataset-specific fluctuations.
4. Add additional scoring metrics like BLEU.

5. Add future costs estimation to avoid translating easy phrases first.
6. Experiment with other reordering algorithms like A* or neural-based ones.

# Conclusion

In this project I implemented two algorithms for solving a problem of phrases reordering: beam search and greedy hill climbing. Also, I added several improvements for vanilla beam search: reordering limits, stack threshold pruning. I conducted careful evaluation of different systems and their modifications and concluded that algorithms, capable of reordering, perform better than monotone decoding algorithms. However, I observe much longer time of running, since advanced algorithms consider much bigger search space for their hypotheses.

## References


1. ARPA Language Model format - https://cmusphinx.github.io/wiki/arpaformat/
2. Koehn slides http://mt-class.org/jhu/slides/lecture-decoding.pdf
3. Basic task description http://mt-class.org/jhu/hw2.html
4. Statistical Machine Translation Koehn book.
5. Langlais, Philippe & Patry, Alexandre & Gotti, Fabrizio. (2007). A greedy decoder for phrase-based statistical machine translation.