# HW2 Writeup: Decoding

**Anjali Narayan-Chen** and **Johnny Chang**
nrynchn2@illinois.edu, jychang3@illinois.edu

## 1 Introduction

For this homework, we adapt the approach of (Langlais et al., 2007) of a local greedy search algorithm. Using a predefined set of operations, the greedy search proceeds by modifying the current-best translation returned by the default stack decoder and searching over the set of modified translations in a hill-climbing approach. If a better-scoring translation is found, the modified translation is returned as the system's prediction.

## 2 Algorithm

Given *source*, a sentence to translate, the core of the greedy search algorithm proceeds as follows:

**Algorithm 1** Greedy Search

```
1:  $current \leftarrow$ seed($source$)
2:  loop
3:      $s\_current \leftarrow$ score($current$)
4:      $s \leftarrow s\_current$
5:      for all $h \in$ neighborhood($current$) do
6:          $c \leftarrow$ score($h$)
7:          if $c > s$ then
8:              $s \leftarrow c$
9:              $best \leftarrow h$
10:     if $s = s\_current$ then return $current$
11:     else
12:         $current \leftarrow best$
```

The idea behind the greedy search is to modify parts of the translation one at a time, exploring the search space of neighboring hypotheses in a hill-climbing fashion. The hope is that such modifications to the sentence output by the original stack decoder will provide search directions that lead to improved hypotheses.

In our implementation, the function `seed` that seeds the search with an initial state is the provided stack decoder. The scoring function, `score`, is similarly defined as the scoring function used in the provided code, $\log p(\mathbf{f}, \mathbf{a}|\mathbf{e}) + \log p(\mathbf{e})$. We next define the `neighborhood` function, which takes a candidate translation as an argument and returns a set of neighboring hypotheses to consider.

### 2.1 Neighborhood Function

The neighborhood function is defined via a set of five operations that can transform a current translation, as defined by (Langlais et al., 2007).

**Swap** The *swap* operation swaps two adjacent target segments. This is designed to combat the baseline model's strong bias toward monotonous translations.

**Replace** Given a specific source segment, the *replace* operation exchanges the translation for that segment with anther found in the phrase translation table.

**Bi-Replace** The *bi-replace* operation works similarly to *replace*, allowing the translation of two adjacent source phrases to change simultaneously. The motivation for this is to modify the sentence enough to escape a possible local maximum in the search.

**Split** The *split* operation splits a given source phrase into two parts and re-translates the split

| Model | Log-Likelihood of Corpus |
|---|---|
| Stack Decoder | -1439.874 |
| Greedy Decoder | -1362.362 |

Table 1: Results of different decoding models.

phrases according to translations found in the phrase translation table.

**Merge** The *merge* operation is the opposite of the *split* operation: it allows two adjacent source phrases to be merged. The merged phrase then receives a new translation according to the phrase translation table.

## 3 Results

Due to the randomness inherent in our *replace* operations, we take an average over five runs of our implementation on the provided corpus. We compare the log likelihood of the corpus under our model against the log likelihood of the corpus returned by the given stack decoder. The results are shown in Table 1.

## References

Philippe Langlais, Alexandre Patry, and Fabrizio Gotti. 2007. A greedy decoder for phrase-based statistical machine translation. *Proc. of TMI*.