Machine Translation Assignment 2: Decoding

s0907677, s1520582

Q1

The default limitation of both stack size and maximum number of translation to 1 is very restrictive and so one expects to see a significant improvement in total log probability once either limit is increased. This proves to be true, but only in a very limited range for both variables, as illustrated in Figure 1.

Changing the maximum number of translations causes a sharp improvement, but it flattens out very quickly. The difference in total log probability between decoder with $k = 10$ and $k = 50$ is only 10% of the difference between $k = 1$ and $k = 2$. No further improvements are observed for $k > 50$.
Changing the stack size has a similar effect, although improvement tapers out even quicker. Increasing stack size to over 3 does not provide significant benefits.
Qualitatively, the difference between translations produced by the maximally limited system ($k = 1$ and $s = 1$) and one which produces the highest scoring output ($k = 50$ and $s = 10$) is that the latter text is more fluent and idiomatic, yet the meaning is equally difficult to grasp and there is no difference between the levels of syntactic correctness.
We did not investigate how parameter changes affect decoding speed. Simple timing methods seemed to be neither sensitive nor reliable enough to uncover any interesting trends for values of k and s in the range [1–100]. However, Figure 2 offers a decoding time comparison with the local reordering case.

The following conclusions can be drawn:

a. The intuitive insight that phrases can have multiple good translations, depending on the context, proves to be important. Increasing the maximum number of possible translations per phrase does improve output quality.

b. The pruning heuristics used in decoding are justifiable, since the best translation can really be found by searching within the top few hypothesis in each stack. The best hypothesis is not always the very top one, as witnessed by the sharp increase in log probability when stack size is changed to 2, but it seems to almost always be one of the top 5 or so.

c. There is a limit, and one that's quickly reached, to how good a monotonic decoder can perform. Allowing it to entertain more hypotheses cannot overcome inherent problems entailed by monotonicity.

Q2

When local reordering is allowed, the search space expands, since at each step we need to consider three possible sources of the next part of the English translation:

- the next French phrase (as was always the case in the monotone decoder);
- the second next French phrase;
- the previous French phrase.

The latter two cases account for the fact that hypotheses might be created by skipping one French phrase and translating a next one. If that was the way a hypothesis was generated, the only way of extending it is to go back and translate the skipped part.
Instead of modelling explicitly these three possibilities, we decided to describe the search space more succinctly by positing that there is only one possible way of expanding a hypothesis, namely finding next two phrases and swapping them. This reformulation is possible thanks to the introduction of an artificial empty phrase, which we call the ε phrase.
In recursive terms, given the last word in the translation, e, and the index of the last translated French word, j, we are looking for such pair of indices i, c that:

– $i < c \leq j$

– translation of French[i,c] ends in word e

– translation of French[c,j] proceeds translation of French[i,c] in the English sentence

– the translation and language model probability is maximized

Given that French[c,j] might be the ε phrase, this formulation also covers the no-reordering case.

Our full definition of $h(j,e)$ is specified as follows:

$$h(0,e) = \begin{cases} 1, \text{if } e = \text{START} \\ 0, \text{otherwise} \end{cases}$$

$$\begin{aligned} h(j,e) = \quad & \underset{h(i,e')e_1 \ldots e_k e_{k+1} \ldots e_m e}{\operatorname{argmax}} && p(h(i,e')) \\ & + && \log p_{\text{TM}}(f_{i+1} \ldots f_c | e_{k+1} \ldots e_m e) \\ & + && \log p_{\text{TM}}(f_{c+1} \ldots f_j | e_1 \ldots e_k) \\ & + && \log p_{\text{LM}}(e_1|e') + \sum_{k'=1}^{m-1} \log p_{\text{LM}}(e_{k'+1}|e_{k'}) + \log p_{\text{LM}}(e|e_m) \end{aligned}$$

with $0 \leq i < c \leq j$, $0 \leq k \leq m$, $e' \in V_E$, $e_1 \ldots e_k \in t(f_{c+1} \ldots f_j) \cup \{\varepsilon\}$,
and $e_{k+1} \ldots e_m e \in t(f_{i+1} \ldots f_c)$.

Q3

For each French sentence of length n, we loop over all n possible stacks. In each stack, we look at the top s hypotheses and expand them. To perform an expansion we choose two consecutive phrases (the second of which might be ε), swap them, and consider all possible combinations of the first k translations of each phrase. The maximum considered length of phrase is n. Finally, the algorithm calculates the language model probability of the generated two-phrase translation. In this step, the algorithm has to iterate over up to $2t$ words, where t is the maximum length of a translation phrase. Thus the overall complexity is as follows:

$$\mathcal{O}(n \cdot s) \cdot \mathcal{O}((n \cdot k)^2) \cdot \mathcal{O}(t) = \mathcal{O}(n^3 \cdot k^2 \cdot s \cdot t)$$

Q4

In our implementation, the mapping from hypotheses to stacks is the same in the decoder with local reordering as it was in the one without. Each hypothesis is placed on a stack corresponding to the index of the last French word being translated. We do not create hypotheses in which French phrases are skipped, but instead always look at two consecutive phrases, swap them, and create a hypothesis which covers both phrases. This means that mapping onto stacks is straightforward, since the number of French words covered by a hypothesis is always the same as the index of the last word which it translates. In short, the hypotheses we generate do not have gaps in coverage of the source sentence and can be simply placed on the stack corresponding to the last word covered.

Q5

Our implementation of local reordering does not involve any changes to the hypotheses objects. We modify the translation model by inserting an empty phrase which translates to English as an empty string with probability 1.
We introduce a modification in the hypothesis extension step. Given a hypothesis which ends at index i in the French sentence we consider all indices c and j such that $c \leq j$, $j \leq |sentence|$, and the fragments of the sentence delimited by *[i, c]* and *[c, j]* are phrases. Since we allow for $c = j$, the second phrase may be the empty one we have added to the translation model. For each discovered pair of phrases we create a hypothesis by swapping them and considering k best translations for each. For a pair with an empty phrase this amounts to no reordering.

Q6

Similarly to the non-reordering decoder, increases in the number of translations k lead to major improvements at first, but fewer improvements $k > 50$. In contrast to the results obtained from the simple decoder, increases in the stack size s do not fade out as quickly. The changes also have a larger effect for smaller s, with the biggest difference between $s = 1$ and $s = 2$. Even in the reordering decoder, we do not observe any performance increases for $s > 20$.
We note that k and s have a much larger effect on runtime than they have for the simple decoder. Nonetheless, the run times of both decoders are in the same order of magnitude (see Fig. 3). For $k = s = 10$, the sentences are relatively well-formed, though still difficult to understand. Since we are still using a bigram model, it is not surprising that long-term dependencies are not correctly resolved.

Q7

We based our decoder on the correspondence between phrase-based decoding and the Traveling Salesman Problem, following the proposal of Zaslavskiy et al. (2009).
Our implementation includes the following steps:

a. project the translation problem to an *Asymmetric Generalized Travelling Salesman Problem* [AGTSP].

b. convert the AGTSP to an *Asymmetric Travelling Salesman Problem* [ATSP].

c. find the best path by utilizing the LKH package[1] implementation of the Lin-Kernighan heuristic (Helsgaun (2006)).

We transform a sentence into an asymmetric graph by following the procedure described in Zaslavskiy et al. (2009): We extract all possible phrases from the French sentence. For each phrase

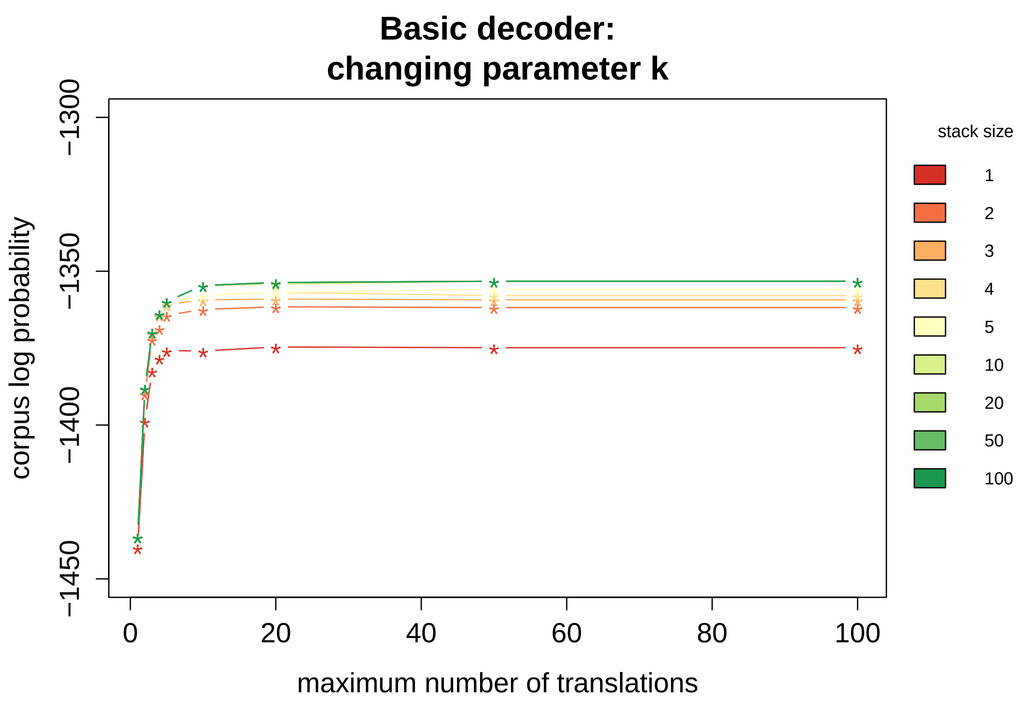
[1]`http://www.akira.ruc.dk/~keld/research/LKH/`

we retrieve k possible translations and store each pair as a bi-phrase. For each combination of a French word and a bi-phrase in which it occurs we create a node (word, bi-phrase) in the AGTSP graph. Nodes sharing the same French word form a group. In AGTSP, each group has to be visited once, which means that the algorithm forces the final translation to cover each French word. We decide on costs for each directed edge following the approach described in the article. Edges connecting nodes which represent consecutive words in the one phrase carry zero costs, whereas the cost of transitions between phrases is determined by the translation model, language model, and the distance between the phrases in the French sentence.
We have to convert our AGTSP to an ATSP so that we can use the solver by Helsgaun (2006). This projection is done in polynomial time and converts each group to a directed cycle of nodes connected with low-cost edges. The exact conversion is described in the article.
The original article optimizes three parameters to weigh the relative importance of translation model, language model, and phrase distance. We simplified the model by setting these parameters to constant 1. We also do no follow the article in further transforming the ATPS graph to a STPS one. Instead we use a solver which implements the same algorithm as the one used in the article, but can operate on the directed ATSPs.
The proposed model does not reach the same translation quality as the default non-swapping decoder, as measured by `compute-model-score`. A possible difficulty lies in the fact that our edge costs are calculated in advance, so that our language model has only a limited context to work on. This is especially a problem for one-word phrases which consist of out-of-dictionary words. A second problem might be that the ATSP solver does not guarantee the optimal solution, although we did not spot any obviously non-optimal solutions in our sample documents. Another, most likely cause of poor performance, might be an error in our implementation of edge cost calculation. There are many factors which influence what kind of an edge connects any two nodes in the graph – are the nodes in the same cluster, do they share a phrase, is any one of them the start-of-sentence-node, etc. In Figure 4, we see the corpus log-probability of our decoder for various numbers of possible translations k.

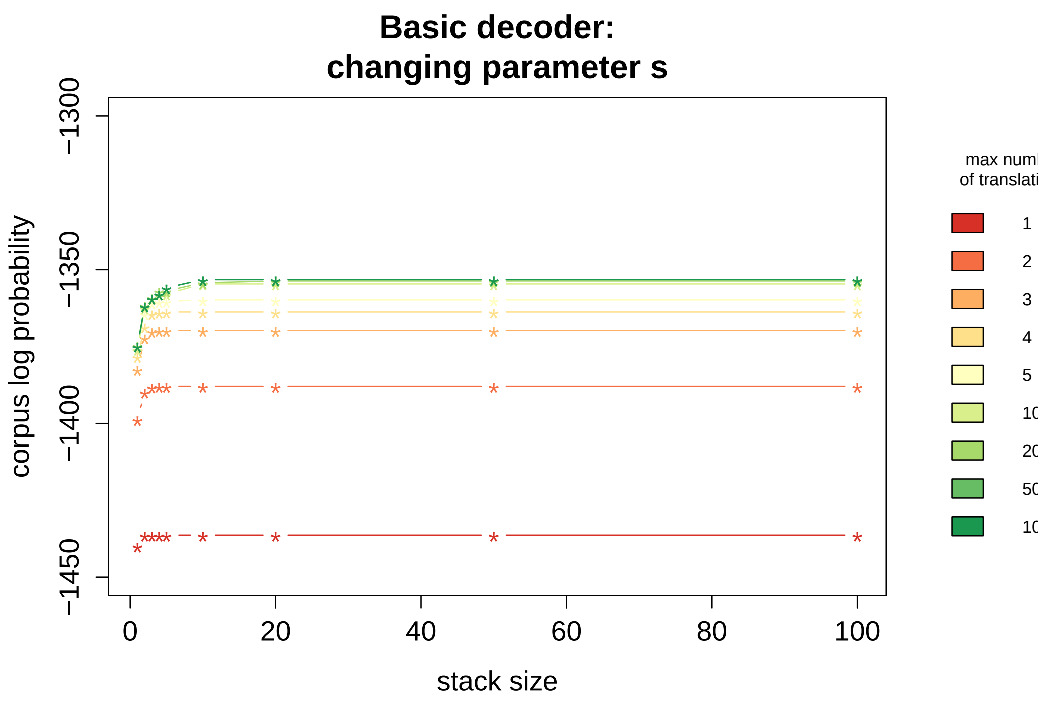
References

Helsgaun, K. (2006). An effective implementation of k-opt moves for the lin-kernighan tsp heuristic. *Datalogiske Skrifter (Writings on Computer Science)*, 109.

Zaslavskiy, M., Dymetman, M., and Cancedda, N. (2009). Phrase-based statistical machine translation as a traveling salesman problem. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 1:333–341.

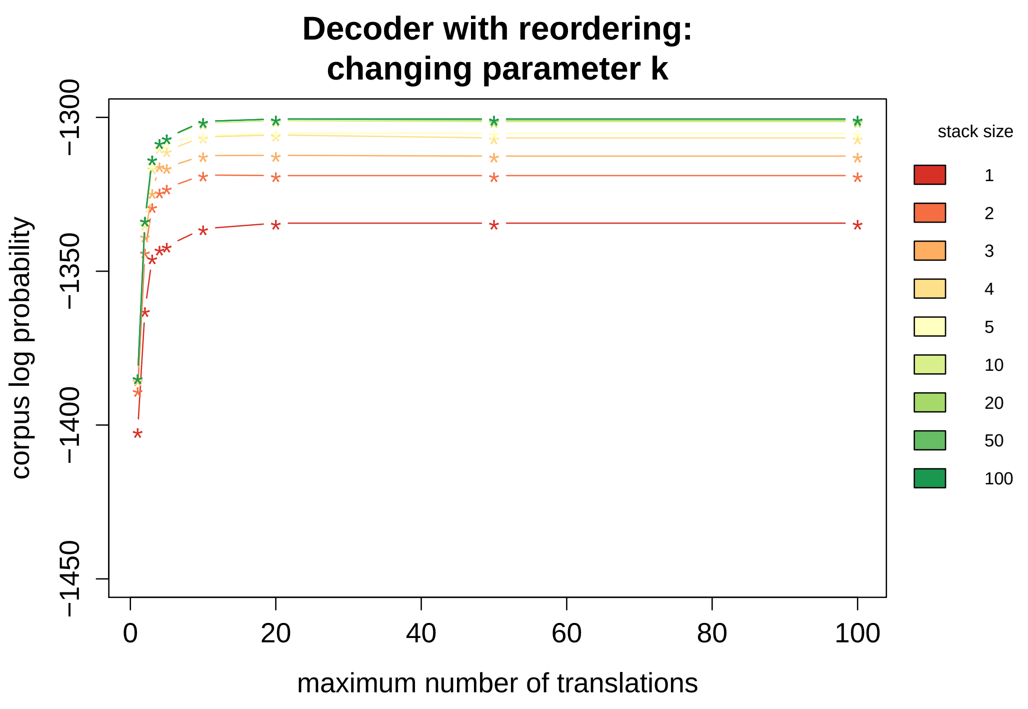


(a) Increasing maximum number of translations, range [1–100].

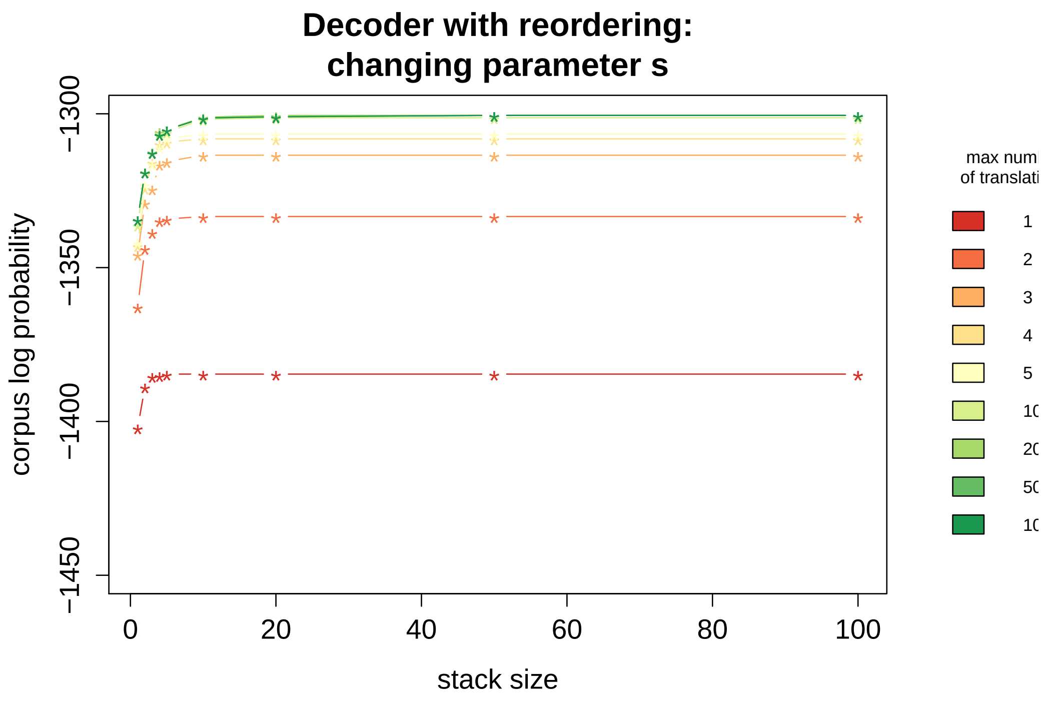


(b) Increasing stack size, range [1–100].

Figure 1: Performance of decoder without reordering.

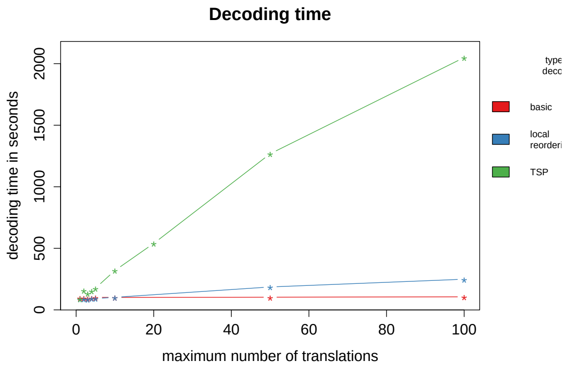


(a) Increasing maximum number of translations, range [1–100].

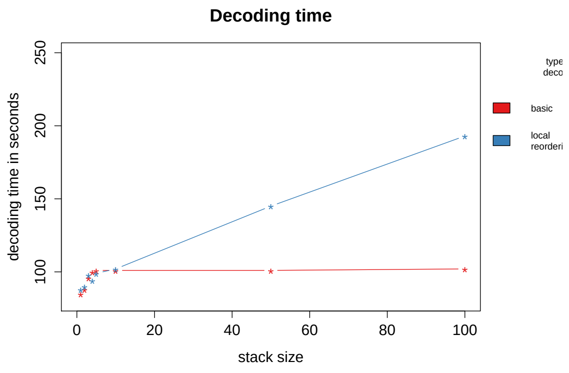


(b) Increasing stack size, range [1–100].

Figure 2: Performance of decoder with local reordering.



(a) Decoding time as function of maximum number of translations.



(b) Decoding time as a function of stack size.

Figure 3: Decoding time for the three decoders.

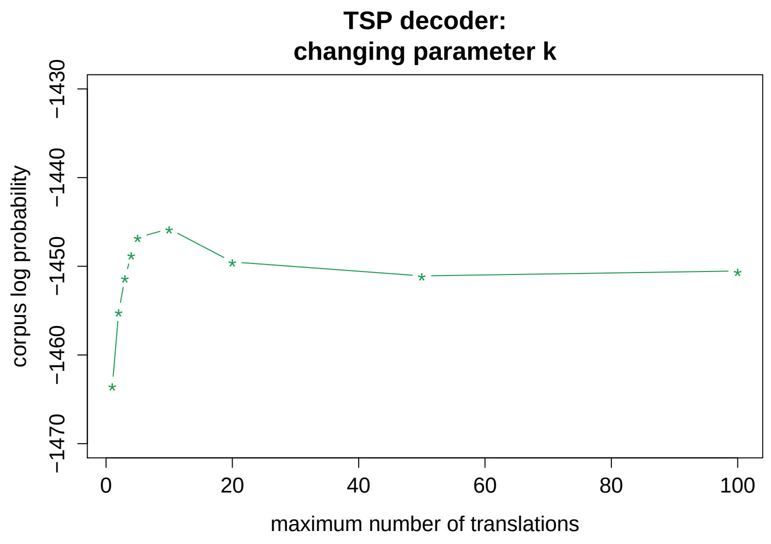


Figure 4: Performance of the TSP decoder as a function of number of translations.