1. (10 points) Experiment with stack pruning parameters. What is their affect on...

a. log-probabilities:
The log-probabilities go up with increases in s and k, though there seems to be no significant increase past either $s > 10$ or $k > 10$. The log-probabilities plateau at around this point. (See figure 1.)
b. speed:
The time seems to go up *at most* linearly with increases in s, though the curve gets steeper with increases in k. Note: in the graphs, both the time and the stack size are plotted logarithmically.
c. translations:
Most of the translations are ungrammatical, and very hard to understand.
d. maximum log-probability:
The log-probability plateaus at -1353.247828.

2. (15 points) Define a new dynamic program for Part 2.

First, let us define some notation for the sum of the log-probabilities of the sequence $e_1 \dots e_k e$ following e' according to the language model:

$$\log p_{\text{LM}}(e_1 \dots e_k e \mid e') = \log p_{\text{LM}}(e_1 \mid e') + \sum_{i=1}^{k-1} \log p_{\text{LM}}(e_{i+1} \mid e_i) + \log p_{\text{LM}}(e \mid e_k)$$

Secondly, we let $h(i, e) = s(0, 0, i, e)$, and we define $s(k, j, i, e)$, for $k = j = 0 \vee k < j$, as follows:

$$s(0,0,0,e) = \begin{cases} \epsilon & \text{if } e = \text{START} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$s(0,0,i,e) = \underset{s(k,j,i',e')e_1 \dots e_n e}{\arg\max} \quad \log p(s(k,j,i',e')) + \log p_{\text{LM}}(e_1 \dots e_n e \mid e') + \begin{cases} \log p_{\text{TM}}(f_{i'+1} \dots f_i \mid e_1 \dots e_n e) & \text{if } k = 0 \wedge j = 0 \\ \log p_{\text{TM}}(f_k \dots f_{j-1} \mid e_1 \dots e_n e) & \text{if } i' = i \end{cases}$$

$$s(k,j,i,e) = \underset{s(0,0,i',e')e_1 \dots e_n e}{\arg\max} \quad \log p(s(0,0,i',e')) + \log p_{\text{LM}}(e_1 \dots e_n e \mid e') + \log p_{\text{TM}}(f_j \dots f_i \mid e_1 \dots e_n e)$$

Note that $s(0, 0, i, e)$ does *not* denote a translation state with a hole in the k^{th} position, as the second argument is the index of the first word *after* the gap. It denotes a translation state without a gap.

3. (5 points) What is the complexity of your Part 2 decoder? Explain your reasoning.

There are 2^{n-1} different segmentations for a sentence of length n. For each of these segmentations, we produce a number of translations by swapping adjacent phrases, equal to the m^{th} Fibonacci number F_m, where m is the number of phrases in the segmentation. Since $F_m = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^m$ is exponential, and there are an exponential number of segmentations, the total number of translations for a sentence is going to be double exponential, i.e. $O(2^{2^n})$, assuming a constant size for the number of translations per phrase. Practically, however, the complexity is further bounded by the stack size used by the beam search implementation.

4. (5 points) What is the mapping from hypothesis objects to stacks for Part 2?

Hypothesis objects are mapped to stacks as $s(k, j, i, e) \mapsto i - (k - j)$, as we need to subtract the number of words in the gap $(k - j)$ from the number of words covered i. Since $h(i, e) = s(0, 0, i, e)$, $h(i, e) \mapsto i$ as in the previous model.

6. (15 points) Experiment with stack pruning parameters for Part 2. What is their affect on...

a. log-probabilities:
The log-probabilities go up with increases in s and k, though there seems to be no significant increase past $s > 50$ and $k > 10$. The log-probabilities plateau at this point. (See figure 2.)
b. speed: The time seems to go up *at most* linearly with increases in s, though the curve gets steeper with increases in k.

c. translations:
The good news is that we get "selection committee" now! There are some other minor improvements, such as "you can see you" → "you will see" and "and I will believe" → "and I believe it". Minor shining lights aside, most of it is still incomprehensible and ungrammatical.
d. maximum log-probability:
The log-probability plateaus at −1300.526262.

7. (10 points) Define a new dynamic program for Part 3.

For Part 2, our program had the choice of either translating the next phrase, or leaving a hole—of exactly one phrase—and translating a phrase following that. However, if it chooses to leave a hole, it had to translate the hole immediately after, as a consequence of the swapping constraint. The description of Part 3 implies that *this* program can translate any number of phrases before going back and translating the hole, as long as it never left more than one hole, and never one of more than one phrase.
The definition of Part 3 is very similar to that of Part 2, sharing—for starters—the same auxiliary definition and the same basic form of $s(k, j, i, e)$, which I will not repeat here. The only real difference is in the final equation, in which it is now possible to continue translating phrases while there is a hole.

$$s(0,0,0,e) = \dots$$
$$s(0,0,i,e) = \dots$$
$$s(k,j,i,e) = \underset{s(k',j',i',e')e_1 \dots e_n e}{\arg\max} \log p(s(k',j',i',e')) + \log p_{\text{LM}}(e_1 \dots e_n e \mid e') + \begin{cases} \log p_{\text{TM}}(f_j \dots f_i \mid e_1 \dots e_n e) & \text{if } k' = 0 \wedge j' = 0 \\ \log p_{\text{TM}}(f_k \dots f_{j-1} \mid e_1 \dots e_n e) & \text{if } i' = i \end{cases}$$

8. (5 points) What is the computational complexity of your Part 3 decoder?

There are 2^{n-1} different segmentations for a sentence of length n. For each of these segmentations, we produce a number of translations by swapping phrases, with the restriction that the arcs of swapping cannot cross or nest. We can describe this as a recurrence relation: we choose an arc of length i, and then compute the number of possibilities for the remainder, or $a_0 = 1, a_n = \sum_{0<i\leq n} a_{n-i}$. This works out to be precisely 2^n, once again giving us a double exponential number of translations, i.e. $O(2^{2^n})$.

9. (5 points) What is the mapping from hypothesis objects to stacks for Part 3?

Hypothesis objects are mapped to stacks in the same way as in Part 2.

11. (5 points) What is the maximum log-probability your Part 3 decoder can obtain? What do you conclude?

−1278.680068, with $s = 2000, k = 100$. For further increases of s, the score does not improve, and given the fact that the score does not change between $s = 1000, k = 50$ and $s = 1000, k = 100$ I am guessing that further increases of k will not make a difference either. Therefore, this is likely the best log-probability which my Part 3 decoder can obtain, and is significantly better than the top scores for my Part 2 decoder. However, with a runtime of just under half an hour, s is impractically large at this point. Especially since we can get a score of −1281.046784 at *one-tenth* the runtime using $s = 500, k = 10$.
All in all, our scores tell us that the Part 3 decoder is an improvement over the Part 2 decoder: the difference between the best for Part 3 and Part 2 is about half as big as the difference between Part 2 and Part 1. Looking at the translations, however, it is hard to see an improvement. Some things give the impression of being *more* garbled. Overall, it does not feel like a tremendous improvement.
If we think about it from a linguistic perspective, this makes sense. Our binary swapping decoder would fix some things, such as the common cases of French adjectives. Longer distance movements would still be translated to the wrong position. In our Part 3 decoder, we can only move single phrases. For instance, if we translate from an SOV language to an SVO language, many translations will involve moving a noun phrase (i.e. the object). We would be able to do this *only* if the object, in its entirety, were present in the phrase table. A program allowing a sequence of phrases to be displaced might work better, but also would massively inflate the number of possible translations. A program which uses swaps in the parse tree would perform better, but we cannot just assume we have a parse tree.
(See figure 3. Data for $s > 1000$ not included.)

List of Figures

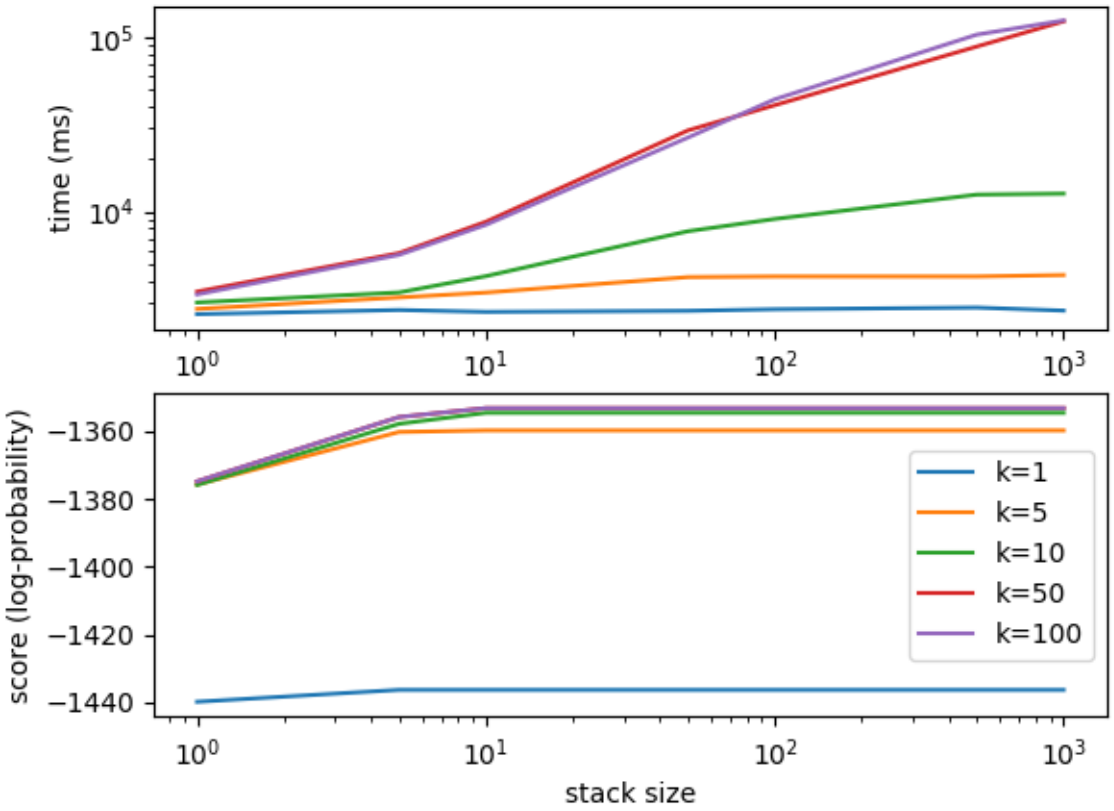

Figure 1: Experiment with stack pruning parameters (baseline).

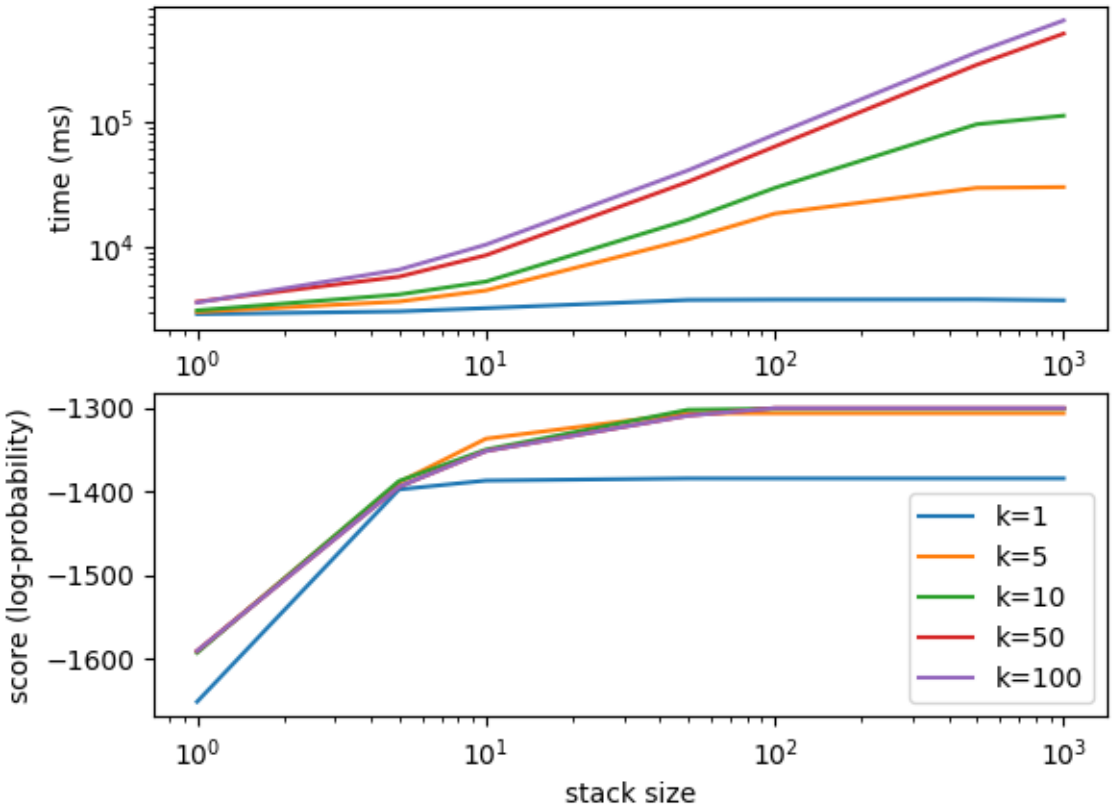

Figure 2: Experiment with stack pruning parameters (part2).

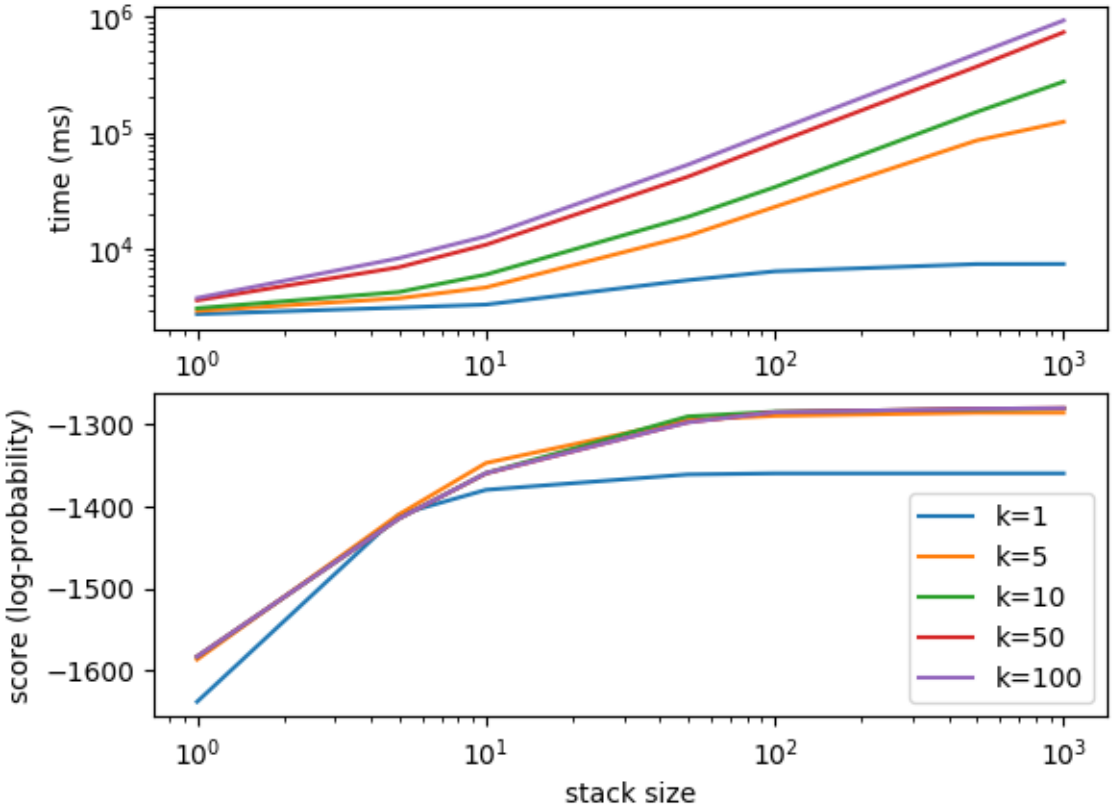

Figure 3: Experiment with stack pruning parameters (part3).