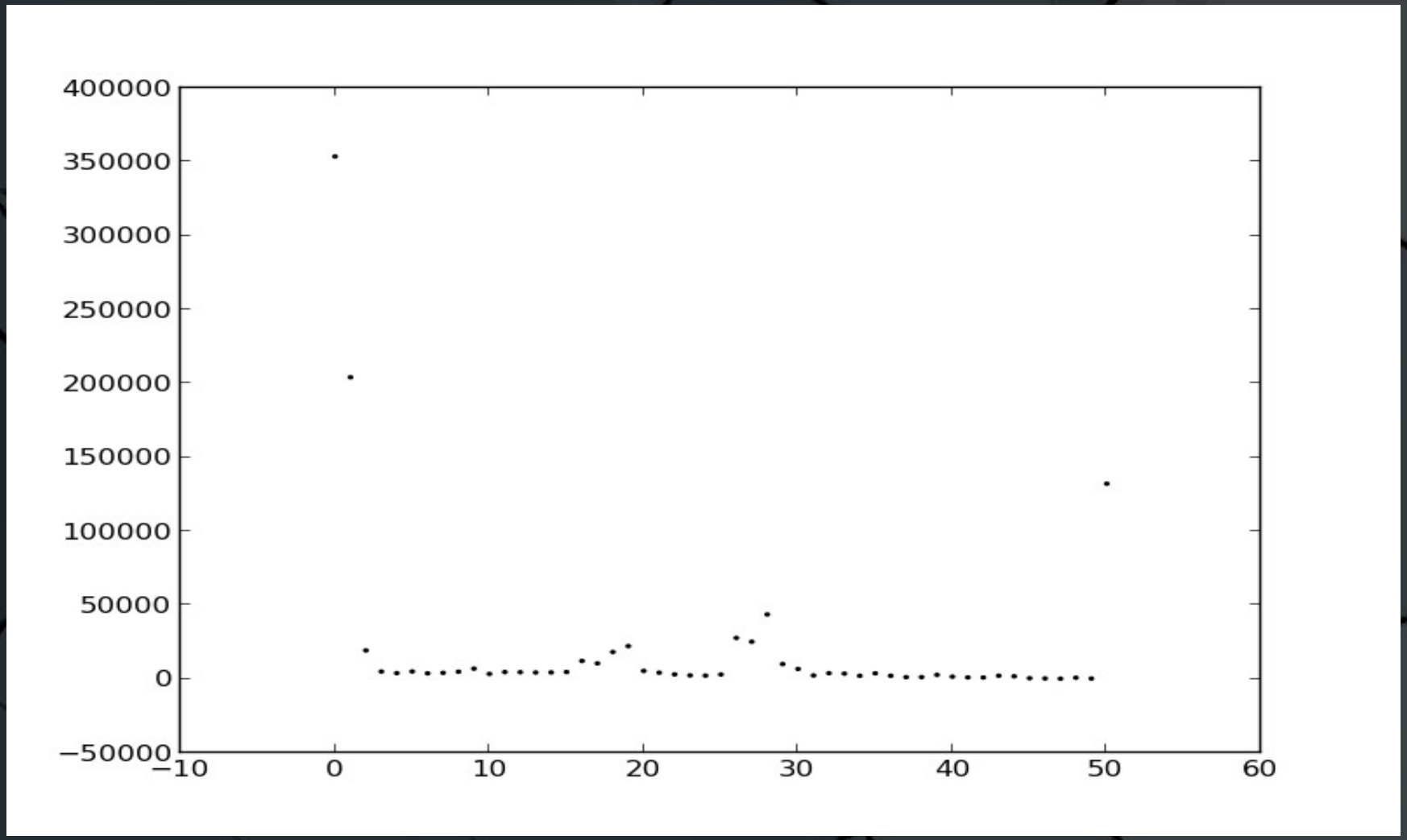
Task 1 – Method and results

- Managed to implement: Brute force in C.
- Wanted to implement: Ukkonen's algorithm.
- Brute force: For every line, iterate over suffixes, if suffix matches adapter, then output suffix.
- Sequences with a suffix matching some prefix of adapter = 646 364
- N = nucleotides per sequence
- L = number of sequences
- Asymptotic worst case runtime = O(L * N^2)
- Practical runtime = 0.47 seconds with an Intel i5 processor

Task 1 – Remaining length distribution



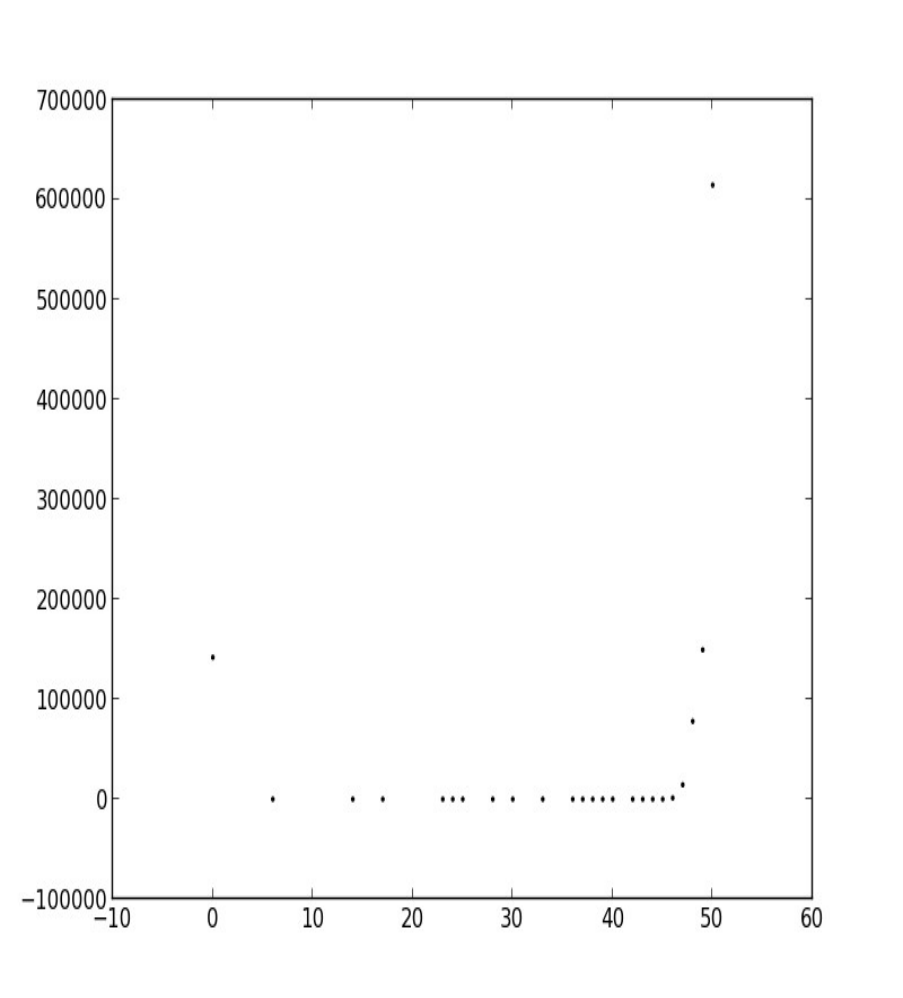
Task 2 – Method and results

- Managed to implement: Brute force in python
- Wanted to implement: l-mer filtration
- Sequences with a suffix matching some prefix of adapter

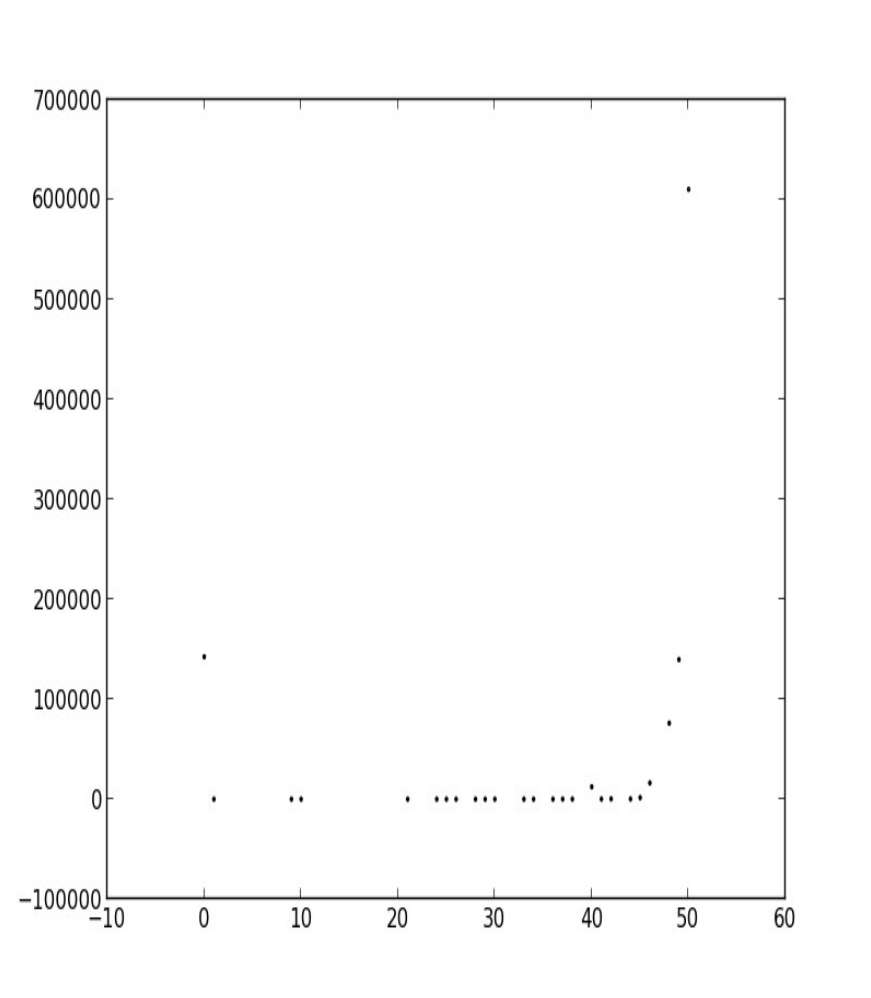
  10% error rate = 385 273. 25% error rate = 389 527
- Asymptotic worst case runtime = O(L * N^2)
- Practical runtime = 140 seconds
-

Task 2 – length distribution

- Remaining sequence length. Error rate 10%



- Remaining sequence length. Error rate 25%



Task 3 – method

- Finally not brute force! (that would have been difficult)
- Managed to implement: Branch and bound motif finding.
- Wanted to implement: Branch and bound motif finding.
- Search space – all possible alignments vs. all possible sequences
- Preprocessing stage – building a hash table of the number of times a suffix is in the DNA Blob. O(lines * line_length). sort of.
- Branch and bound stage – worst case exponential. Average ???
- Optimizations:
  - Keep track of “used up” potential at each index (keeps the upper_bound down)
  - Only generate children that are in the hash table. (legal? Removes some optimal solution?)

Task 3 – results

- Går tom for minne på s_1_sequence_1M
- Men på s_3_sequence_1M
- Practical runtime: 107 seconds, (2.2 GB memory)
- Infered sequence on s_3_sequence_1M

  “GCCTCTGGATGCGGTAATTGTCCCTGAGTGGGAACAGGCACCACCCGCTG”
-

Task 3 – length distrubution