String Search

Helen Garabedian

February 21, 2025

1 Introduction

DNA serves as the fundamental blueprint for every living organism, encoding the genetic information required for growth, development, and function. In genomics research, comparing DNA sequences is essential for linking genetic variations to observable traits or diseases. This task often involves finding a short nucleotide sequence, called the pattern P, within a much larger DNA sequence, referred to as the text T.

This document presents a comparative analysis of two string search algorithms applied to this genomic challenge. The first algorithm, a basic brute-force approach, methodically examines each possible position in T for a match with P, resulting in a runtime that increases with the length of T. The second algorithm, the Boyer–Moore method, employs intelligent heuristics to skip unnecessary comparisons, often achieving faster search times. Our experiments evaluate these algorithms by measuring both execution time and memory usage, thereby highlighting the practical trade-offs between algorithmic simplicity and efficiency in computational genomics.

2 Results

Figure 1 shows the empirical performance of the naive and Boyer–Moore search algorithms. The top panel illustrates the average runtime (in nanoseconds) as the text size increases from 100 to 10,000 characters, while the bottom panel shows the corresponding memory usage (in bytes).

The results indicate that the Boyer–Moore algorithm generally outperforms the naive search in terms of runtime, particularly for larger texts, while the memory usage for both algorithms remains comparable.

3 Methods

3.1 Experimental Setup

We evaluated the performance of the naive and Boyer–Moore search algorithms by measuring runtime (in nanoseconds) and memory usage (in bytes) under varying text sizes while keeping the pattern size fixed. The specific parameters used in the experiments are as follows:

- **Text Size:** Ranged from 100 to 10,000 characters (in increments of 100).
- **Pattern Size:** Fixed at 100 characters.
- **Rounds:** Each experiment was repeated 5 times to compute average performance metrics.
- **Data Generation:** Random text was generated using the alphabet $\{A, C, T, G\}$ to simulate DNA sequences. For each experiment, a random substring of the generated text was selected as the pattern.

3.2 Implementation Details

The experiments were implemented in Python. The naive search algorithm is defined in `naive_search.py`, and the Boyer–Moore algorithm is implemented in `boyer_moore.py`. A common test harness in `string_search.py` was used to:

1. Generate random strings and corresponding patterns.

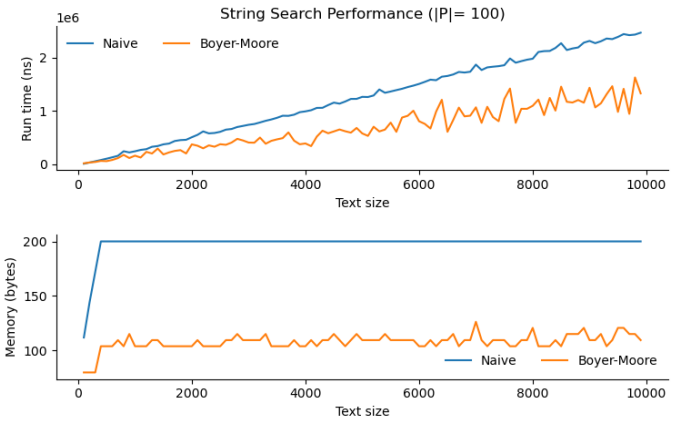

Figure 1: Empirical comparison of the native and Boyer-Moore search algorithms. The top pannel shows the average runtime (in nanoseconds) as the text size increases from 100 to 10,000, with a fixed pattern size of 100. The bottom pannel shows the corresponding memory usage. Boyer-Moore generally outperforms native search in runtime while using comparable amounts of memory

2. Run both search algorithms to measure runtime and memory usage.

3. Plot the results for visual comparison.

3.3 Reproducibility

To replicate these experiments, clone the repository and then run the following commands from the root directory of the repository.

```
$ git clone https://github.com/cu-compg-spring-2025/assignment-4-string-search-helengarabedian.git
$ cd assignment-4-string-search-helengarabedian
$ python src/string_search.py \
  --text_range 100 10000 100 \
  --pattern_size 100 \
  --rounds 5 \
  --out_file doc/combined_search.png
```