Project Proposal

CSCI 6105 Algorithm Engineering

Balaji Sukumaran (B00948977) bl664064@dal.ca

Project Topic:

Development and Testing of a Multipattern Search Algorithm to determine the sentiment polarity of multiple documents efficiently.

Project Motivation:

Imagine an algorithm that goes through the text just once, searching for all the patterns in the dictionary simultaneously. which is designed to offer the best complexity for any input and doesn't require much additional memory [2]. Aho-Corasick search [1], is one of the most effective algorithms for searching patterns in a large text volume. This optimal string search algorithm was invented by Alfred V. Aho and Margaret J. Corasick.

The set of pattern strings is referred to as a dictionary. We denote the total length of its constituent strings as ***m*** and the size of the alphabet as ***k***. The algorithm constructs a finite state automaton based on a trie in **O(*mk*)** time and then processes the text using it [1].

Aho-Corasick search has numerous real-world applications. For example, it is used for exact match and fuzzy matching in network intrusion detection systems [3]. Due to its simplicity and caching properties, parallel and hardware-software co-design implementations of the Aho-Corasick algorithm are utilized in DNA sequence matching, protein identification, and smart IoT [4][5].

In this project, I intend to build a document polarity determination library with the Aho-Corasick search (a multiple-pattern matching algorithm) as its core for efficient classification and compare its efficiency with Knuth-Morris-Pratt (a single-pattern matching algorithm) implementation of the same.

Hypothesis and expected results:

To determine the polarity, I will use a bag of stop [8], negative and positive words [6][7] and employ the Aho-Corasick search algorithm to scan documents in a directory, using these negative and positive words as patterns. This approach will be compared against the performance of a single pattern matching algorithm, Knuth-Morris-Pratt (KMP) [9], in terms of the following criteria:

CPU Utilization – Since the Aho-Corasick search preprocesses all patterns at once, it is expected to use fewer CPU resources compared to KMP.

Memory Utilization – The KMP algorithm is believed to use less memory than Aho-Corasick since it discards the prefix table and constructs a new one for each subsequent search.

Performance Improvement - The Aho-Corasick search is anticipated to show a drastic performance improvement as it can search for multiple patterns in linear time.

I will run the module on multiple documents with varying polarities. The results are expected to be identical in both algorithms, but the Aho-Corasick implementation is projected to retrieve similar results much faster.

In order to profile the algorithm's resource usage, I will be using the Tracy profiling tool [18] to log the memory and CPU utilization at a nanosecond level.

Literature Review:

Why is the multi-pattern search algorithm important?

There have been several instances where the direct application of the Aho-Corasick algorithm has led to drastic improvements in performance and reduced memory utilization. T. Kida et al. [10] addressed the issue of searching directly in LZW-compressed text and presented a new algorithm for finding multiple patterns by simulating a move to Aho-Corasick. They implemented a simple version of the algorithm and demonstrated that it is approximately twice as fast as decompressing and then searching using the Aho-Corasick machine. Yang et al. [11] proposed a Pipelined Affix Search with Tail Acceleration (PASTA) architecture for solving dictionary-based string matching in network deep packet inspection. This architecture, constructed with Aho-Corasick, achieved performance surpassing the requirements of next-generation security gateways for deep packet inspection.

Kennedy et al. [12] introduced a new multi-pattern matching algorithm capable of searching for fixed strings contained within these rules at a guaranteed rate of one character per cycle, regardless of the number of strings or their length. This algorithm, based on the Aho-Corasick string matching algorithm with modifications, resulted in a memory reduction of over 98% on the strings tested from the Snort ruleset.

Guiding paper:

T. Tao et al. [15] claim that their multiple-pattern matching algorithm is practically the fastest among all approaches when the number of patterns is not very large. The proposed algorithm is efficient for large files and is particularly effective when applied to archival searches, especially if the archives are compressed using a common LZW Trie. This claim was validated by comparing it against Kida's Algorithm's performance. I will use a similar approach to prove the performance and resource utilization improvements of the Aho-Corasick search by comparing against KMP algorithm.

Data Sources:

1. ***For patterns -*** I will use the stop [8], positive and negative word sets [6][7].
2. ***Dictionary*** - I will use the free eBooks from Project Gutenberg for sentiment analysis [13]. It has a public-facing API, Gutendex, which provides ease of access for downloading the eBooks [14].

Rough Project Plan:

I plan to implement the following algorithmic techniques:

Modeling - Construct the trie data structure with suffix link and final link as a graph using the NetworkX [17] Python library.

Implementation – Build a sentiment analysis module using the Aho-Corasick search. Each dataset will be going through the following process:

> **Step #1**: Construct Trie data structures with suffix and final links for stop, positive and negative words.
> **Step #2**: For each word try to find it in stop, positive and negative Trie.
> **Step #3**: if the word is found in
>
> - *Stop Trie*: move to the next word

- *Positive Trie*: increment the polarity by 1 and move to the next word
- *Negative Trie*: decrement the polarity by 1 and move to the next word

Step #4: Return the document polarity.

Optional Requirement: Modify the Notepad++ codebase [16] to include this module as a feature.

Experiments - Compare the performance of the Aho-Corasick search and KMP algorithm in terms of space, time, and resource utilization.

Milestones:

Programming Language Constraints: As I am new to Python and C++, I need to enhance my skills in these languages to write production-ready code that flawlessly implement the algorithm.

Complex Implementation: I have to understand and implement Aho-Corasick search algorithm.

References:

[1] "Aho-Corasick algorithm", *Algorithms for Competitive Programming (cp-algorithms.com)*, Available: https://cp-algorithms.com/string/aho_corasick.html, [Accessed: Jan 21, 2024].

[2] S. Subramanian, "Aho Corasick Algorithm", *Medium*, [Apr 13, 2021], Available: https://medium.com/pattern-searching-algorithm/aho-corasick-algorithm-7e5c2e58861c, [Accessed: Jan 21, 2024].

[3] M. Karimov, K. Tashev and S. Rustamova, "Application of the Aho-Corasick algorithm to create a network intrusion detection system," *2020 International Conference on Information Science and Communications Technologies (ICISCT)*, Tashkent, Uzbekistan, 2020, pp. 1-5, doi: 10.1109/ICISCT50599.2020.9351435.

[4] D. R. V. L. B. Thambawita, R. G. Ragel and D. Elkaduwe, "An optimized Parallel Failure-less Aho-Corasick algorithm for DNA sequence matching," *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, Galle, Sri Lanka, 2016, pp. 1-6, doi: 10.1109/ICIAFS.2016.7946533.

[5] S. M. Vidanagamachchi, S. D. Dewasurendra and R. G. Ragel, "Hardware software co-design of the Aho-Corasick algorithm: Scalable for protein identification?," *2013 IEEE 8th International Conference on Industrial and Information Systems*, Peradeniya, Sri Lanka, 2013, pp. 321-325, doi: 10.1109/ICIInfS.2013.6732003.

[6] Mukul, "Positive and Negative Word List.rar", *Kaggle,* Available: Positive and Negative Word List.rar (kaggle.com), [Accessed: Jan 21, 2024].

[7] B. Liu and M. Hu , "Opinion Mining, Sentiment Analysis, and Opinion Spam Detection", *University of Illinois Chicago,* Available: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html, [Accessed: Feb 09, 2024].

[8] R. Swami, "All English Stopwords (700+)", *Kaggle,* Available: All English Stopwords (700+), [Accessed: Feb 09, 2024].

[9] “Algorithms in Bioinformatics”, Available: kmp.pdf (ubc.ca), [Accessed: Jan 21, 2024].

[10] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki and S. Arikawa, "Multiple pattern matching in LZW compressed text," *Proceedings DCC '98 Data Compression Conference (Cat. No.98TB100225)*, Snowbird, UT, USA, 1998, pp. 103-112, doi: 10.1109/DCC.1998.672136.

[11] Y. -H. E. Yang, H. Le and V. K. Prasanna, "High Performance Dictionary-Based String Matching for Deep Packet Inspection," *2010 Proceedings IEEE INFOCOM*, San Diego, CA, USA, 2010, pp. 1-5, doi: 10.1109/INFCOM.2010.5462268.

[12] A. Kennedy, X. Wang, Z. Liu and B. Liu, "Ultra-high throughput string matching for Deep Packet Inspection," *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, Dresden, Germany, 2010, pp. 399-404, doi: 10.1109/DATE.2010.5457172.

[13] “Project Gutenberg is a library of over 70,000 free eBooks”, *Project Gutenberg,* Available: https://www.gutenberg.org/, [Accessed: Jan 21, 2024].

[14] “JSON web API for Project Gutenberg ebook metadata”, *Project Gutenberg,* Available: Gutendex, [Accessed: Feb 09, 2024].

[15] Tao Tao and A. Mukherjee, "Multiple-pattern matching for LZW compressed files," *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, Las Vegas, NV, USA, 2005, pp. 91-96 Vol. 1, doi: 10.1109/ITCC.2005.206.

[16] “Notepad ++”, *Github,* Available: https://github.com/notepad-plus-plus/notepad-plus-plus, [Accessed: Jan 21, 2024].

[17] “Network-X”, *Network-x,* Available: NetworkX — NetworkX documentation [Accessed: Jan 21, 2024].

[18] “Tracy Profiler”, *Tracy Profiler*, Available: Tracy Profiler | Flax Documentation (flaxengine.com) [Accessed: Feb 09, 2024].