行政院國家科學委員會專題研究計畫　期中進度報告

子計畫三:網路內容分類的系統架構：設計、實作與評估 (1/3)

計畫類別： 整合型計畫
計畫編號： NSC93-2213-E-009-112-
執行期間： 93 年 08 月 01 日至 94 年 07 月 31 日
執行單位： 國立交通大學資訊科學學系(所)

計畫主持人： 林盈達

報告類型： 精簡報告
報告附件： 出席國際會議研究心得報告及發表論文
處理方式： 本計畫可公開查詢

中　華　民　國 94 年 5 月 27 日

一，計畫

計畫名稱	以內容為基礎之網路安全-子計畫三:網路內容分類的系統架構：設計、實作與評估(1/3)
計畫編號	93-2213-E-009-112-
主持人	林盈達 教授 交通大學資訊科學系
執行機關	交通大學資訊科學系
執行期限	2004.08.01 至 2005.07.31

二、關鍵詞

本文關鍵詞— 字串比對，模式比對演算法，內容安全
Keywords—string matching, pattern matching algorithm, content security

三、中英文摘要

在網際網路中，網路內容安全已經成為大眾所關切的議題。而字串比對演算法的效率問題是內容處理的關鍵也漸漸被證實。因此我們剖析一些典型的演算法來了解各種演算法適合使用在什麼情況。在剖析的過程中，我們從中觀察出一些實作的設計議題，諸如選擇適當的雜湊函數與提出新的 CRKBT 演算法來改善舊有演算法的效能。改善後的演算法在遇到大量字串模式的時候，效能能夠比舊的方法快 4 倍。我們並選擇三種內容安全套件來實驗並把各別最有效的演算法實作到這三個套件中。改良後的套件效能皆有重大突破。例如在 ClamAV 套件中，新的方法可以比舊的方法快 5 倍以上。最後，我們亦觀察在真實的套件中處理真實資料與人造資料有哪些差異。在真實套件中，套件的處理時間會受到由特徵值所產生的字元集的分布與英文字母的特性而與人造的資料有所差異。

Network content security has become a critical issue for the Internet. It is also proved that the efficiency of the string matching algorithm is essential to content processing. We profile some typical algorithms to understand which algorithm is suitable on what situation. In the profiling process, we find out some practical design issues, such as selecting a best hash function, and propose the CRKBT algorithm to improve the performance of the RKBT algorithm. Its performance is four times faster than the original algorithm for a huge pattern set. We select three content security applications to experiment and implement the most efficient algorithms into them. The performance has significant improvement. For example, the novel method is five times faster than the original method on the ClamAV package. In addition, we also observe the difference between the real and synthetic data on the packages. The processing time on real applications is sensitive to the character set distribution generated from the multiple patterns, signatures and content keywords, and the property of the English words.

四、計畫目的

No existing string matching algorithms can scan signatures of various characteristics more efficient than others. For example, the Wu-Manber algorithm [1] is not efficient for a huge pattern set [2]. Furthermore, all the above applications have signatures of different characteristics. For example, the anti-virus applications have a large number of signatures, and the intrusion detection systems have short patterns of one or two characters. This work investigates the types of signatures in these content security applications and the type that each string matching algorithm can scan most efficiently, and hence the most efficient algorithm is derived for each application.

The efficiency of six typical string matching algorithms are profiled for signature sets varying in sizes, the minimum length of the signature in them and the character set that the signatures are composed of. Sample sets of both synthetic and real signatures are studied to see if there are deviations in the profiling results for both cases. The edges and limitations of each algorithm are better understood after the profiling. The impacts on performance of memory and cache accesses are also measured quantitatively.

The contributions are summarized as follows:

- Finding out the most efficient algorithm for each application of content security
- Proposing the Classified RKBT algorithm to enhance the performance of the original RKBT algorithm.
- Comparing performance for synthetic and real data in these algorithms.

五、研究方法及結果

I. Selected Algorithms

This work categorizes string matching algorithms into four major approaches according to the design philosophy and the data structure that drives the matching: automaton-based, heuristic-based, hashing-based and bit-parallelism-based. An automaton-based approach tracks the partial match of the pattern prefixes in the text. A heuristic-based approach relies on one or two heuristic functions to look up the shift distance. If the shift distance is 0, a verification algorithm is needed to verify if a true match occurs. A hashing-based approach checks a possible appearance of the patterns by hashing a block of characters in the text and compares the hash value with those from hashing the blocks in the patterns. A bit-parallelism-based approach takes advantage of the parallelism of the bit operations inside a computer word to simulate the operation of a finite automaton [3]. Table 1 compares the typical algorithms in these four categories.

TABLE 1: Classification of typical algorithms.

Algorithms	Approach	Time Complexity	Search Type
Aho-Corasick	Automaton-based	Linear	Prefix
Optimized-AC	Automaton-based	Linear	Prefix
Boyer-Moore	Heuristic-based	Sub-linear	Suffix
Horspool	Heuristic-based	Sub-linear	Suffix
Set-wise BMH	Heuristic-based	Sub-linear	Suffix
Wu-Manber	Heuristic-based	Sub-linear	Suffix
Modified-WM	Heuristic-based	Sub-linear	Suffix
Rabin-Karp	Hashing-based	Linear	Prefix
RKBT	Hashing-based	Linear	Prefix
FNP	Hashing-based	Sub-linear	Prefix
SOG	Bit-parallelism-based	Linear	Prefix
BG	Bit-parallelism-based	Sub-linear	Factor

We also select open source packages for observation and experiments in the profiling because the source code is available. They are Snort for IDS, DansGuardian for Web filtering and ClamAV for virus scanning.

II. Practical Design issues

We first use the RKBT algorithm as the searching algorithm and the details of the RKBT algorithm is showed on Figure 1. The detailed description about the RKBT algorithm is as follows. At pre-processing time, the sorted 32-bit hash table is constructed by the first hash value, which is formed of the first four bytes of the pattern, and the second hash is calculated from the first one by xor'ing together the lower 16 bits and the upper 16 bits. Then the second hash is used to build a 2^{16} bitmap. For example, the *i*'th bit is one, if there is at least one pattern with *i* as its second hash value, and zero, if no pattern has *i* as its second hash value. At searching time, first stage, the program can quickly check from the 2^{16} bitmap. When the second hash value is one, the bit 2345 on figure 1, it will run into second stage, and it uses the binary search method on the 32 bit hash table in order to do further inspection. If the hash value is found, it will compare the text with the candidate pattern.

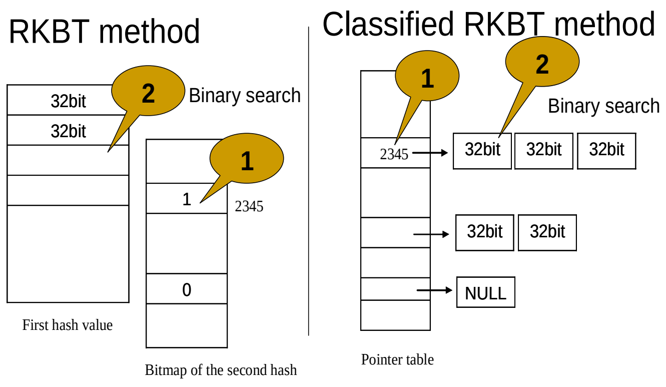

FIGURE 1: the RKBT algorithm vs. the Classified RKBT algorithm

As the number of pattern grows huge, the probability of one occurrence on the second hash table will increase, and consequently it will consume the most time in binary search. According to this observation, we propose the classified approach to improve the original RKBT algorithm, named as CRKBT. The CRKBT algorithm also used two-level hashing and the binary search method. But the data structure and the steps under searching stage are not the same. The CRKBT algorithm uses the pointer table instead of the bitmap table. For instance, the *i*'th pointer point to fixed sorted array, which is constructed at least one pattern with *i* as its second hash value, and point to NULL, if no pattern has *i* as its second hash value. At searching time, the program needs to check the pointer table first. While the pointer exists on specific slot, it will run binary search on this specific sorted array. The search scope will be reduced to a small subset that has the same second hash value. The efficiency of this algorithm will continue beyond 100,000 patterns.

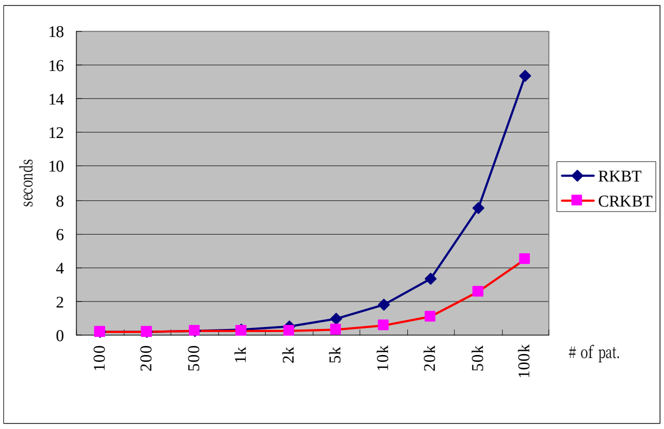

FIGURE 2: the RKBT algorithm vs. the CRKBT algorithm

Figure 2 shows the experimental results. When the number of pattern is small, both algorithms are no need to run into verification stage. The execution time of both algorithms is the same. As the number of pattern increases gradually, it is increasing that both algorithms run into verification stage. But the search scope of the CRKBT algorithm is smaller than that of the RKBT algorithm, the performance gain of the CRKBT algorithm will appear and the CRKBT algorithm is faster than the RKBT algorithm. For example, the CRKBT algorithm is four times faster than the RKBT algorithm when the number of pattern grows to 100,000 patterns. The CRKBT algorithm is more efficient than the other and suitable for huge pattern sets.

III. Profiling Algorithms

We first implement some earlier algorithms, the AC and WM algorithm, and some novel algorithms, the BG and SOG algorithm, and test them to get average time over 1,000 runs using the same text and patterns. Moreover, the implementation of the WM algorithm refers to that of the agrep package and the WM algorithm discussed below generally points at the implementation of the agrep package.

Figure 3 shows the benchmarking results which are tested with LSP=8. We also compare with the CRKBT algorithm. This experiment demonstrates that the Modified-WM algorithm is more efficient than the others when the pattern set size is smaller than 20,000. However, when the pattern set size is greater than 20,000, the CRKBT algorithm is the most efficient. The Modified-WM algorithm and CRKBT algorithm are the fastest ones, so we select these two algorithms as traditional algorithms and compare them with two novel algorithms, the BG+ and SOG+ algorithms.

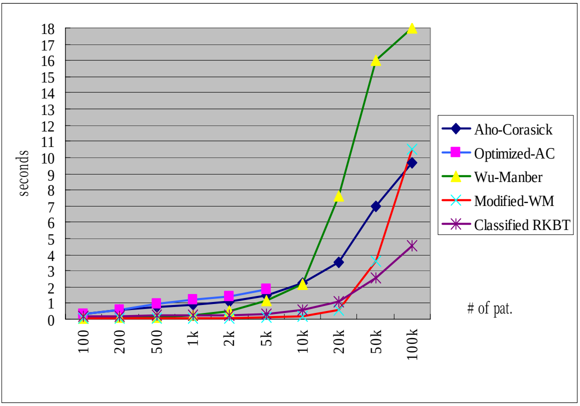

FIGURE 3 Earlier Algorithms Benchmarking Results

Figure 4 shows the benchmarking results which are also tested with LSP=8. We can find out the traditional algorithms are less efficient than the others. The 2-gram BG+ algorithm is the fastest one than the others when the pattern set size is smaller than 50,000. As the pattern set size is greater than 50,000, the 3-gram BG+ algorithm is the fastest one.

In the experiments of earlier and novel algorithms, we can conclude the BG+ algorithm is the more efficient algorithm than the others for LSP=8. As to verify the benchmarking results, we will do internal profiling later.

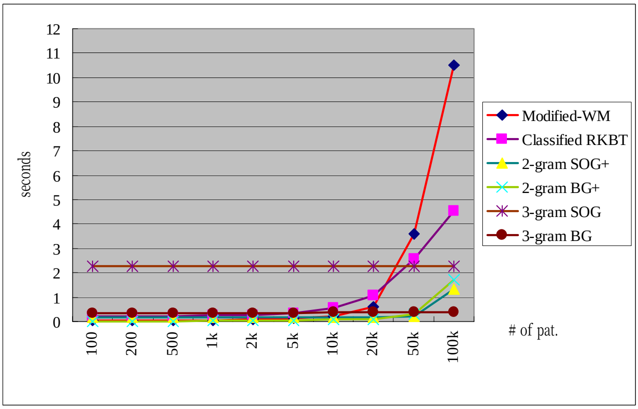

FIGURE 4 Novel Algorithm Benchmarking Results

After the external profiling, we can know what algorithm is the most efficient. But some results need to verify. For example, why does the BG+ algorithm has good efficiency and the Modified-WM algorithm is more efficient than the WM algorithm? We will go through the internal profiling so as to answer the questions. The shift distance, the potential matching and the memory accesses of each algorithm are profiling as follows.

Both the WM and BG+ algorithms are the sub-linear ones. The WM algorithm uses the shift table to record the shift value. The BG+ algorithm also uses the B table plus the bit-parallelism method to calculate the shift value, where the B table keeps whether each character of all patterns occurs or not. So we will profile the shift distance in order to justify the prior results.

Figure 5 shows the profiling results of the average shift distance. According to the results of the average shift distance, we can find out the average shift distance of the WM algorithm is close to one character when the pattern set size between 5,000 and 100,000. So the WM algorithm is not suitable for huge pattern sets. The average shift distance of the Modified-WM algorithm is greater than that of the WM algorithm. This result easily proves the Modified-WM algorithm is more efficient than the WM algorithm. In addition, it is clearly proved that the 2-gram BG+ algorithm is more efficiency when the pattern set size is smaller than 20,000 and the 3-gram BG+ algorithm has larger shift distance than 2-gram BG+ algorithm while the pattern set size between 20,000 and 100,000.

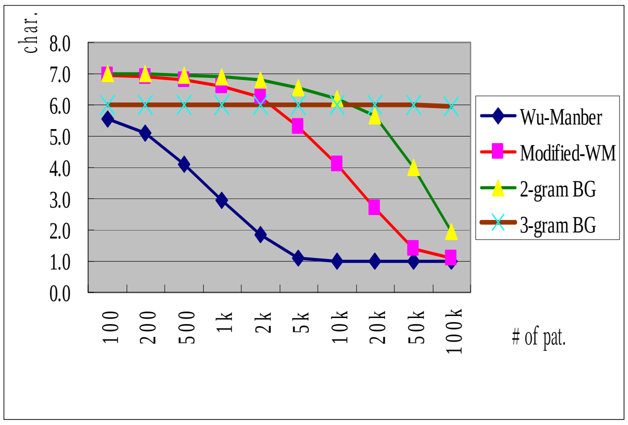

FIGURE 5 the Shift Distance Profiling

Some algorithms are filtering ones that need the verification algorithm to check whether the potential match is a true match or not. As the number of potential matches increase, the string matching performance will decrease and the verification become a bottleneck. The number of the potential matches will be profiled in each filtering algorithm in this section.

Figure 6 shows the percentage of the potential matching for all filtering algorithms. The result shows the potential matching of the Modified-WM algorithm is less than that of the WM algorithm. This result proves the Modified-WM algorithm is more efficient than the WM algorithm, too. In addition, the thing that the potential matching of the WM algorithm increases fast also proves the WM algorithm is less efficiency while the pattern set size is more than 10,000. Finally, it is also proved that the BG+ algorithm is more efficient than the Modified-WM algorithm.

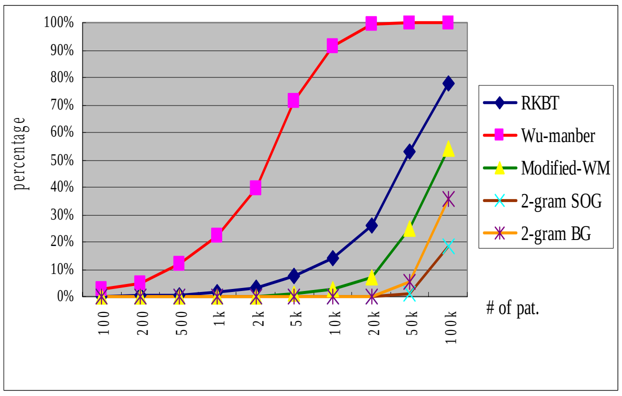

FIGURE 6 the Potential Matching Profiling

Figure 7, 8 and 9 show the results of the number of total memory accesses from program level profiled from Valgrind [4]. The memory accesses of these three figures are the same as well as the results of the external profiling, because the properties of these three types of algorithms are the same. For example, the RKBT and CRKBT algorithms have the same hash function, hash table size and cache miss rate. In addition, it is also proved here again that the CRKBT algorithm is more efficient than the RKBT algorithm.

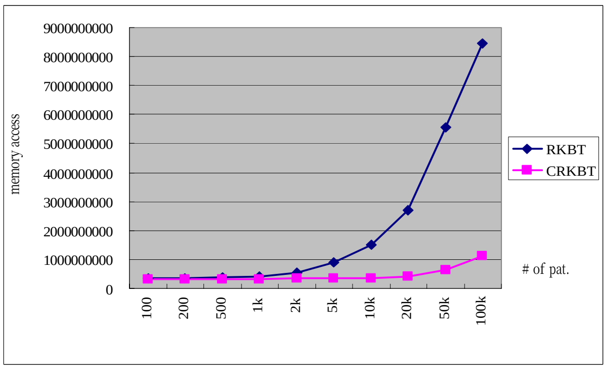

FIGURE 7 RKBT vs. CRKBT

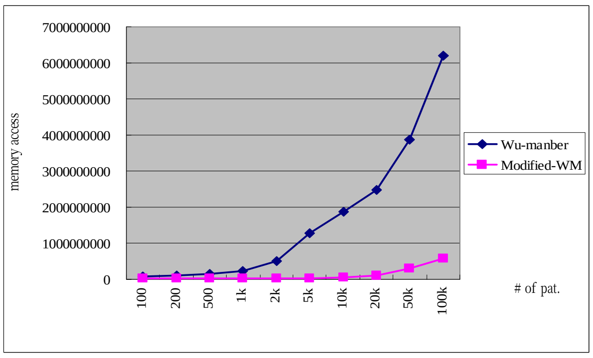

FIGURE 8 Wu-Manber vs. Modified-WM

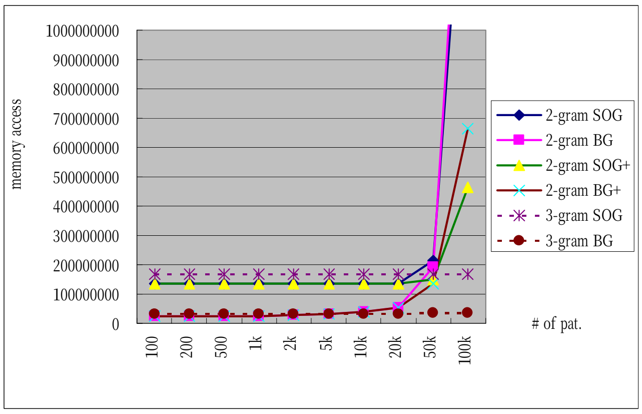

FIGURE 9 BG+ vs. SOG+

When the algorithms of different properties are compared with each other, the results are not the same as above under the huge pattern sets. Because huge pattern sets can bring about many verifications and the cache miss rate are not similar to each other. This result can be observed on figure 10.

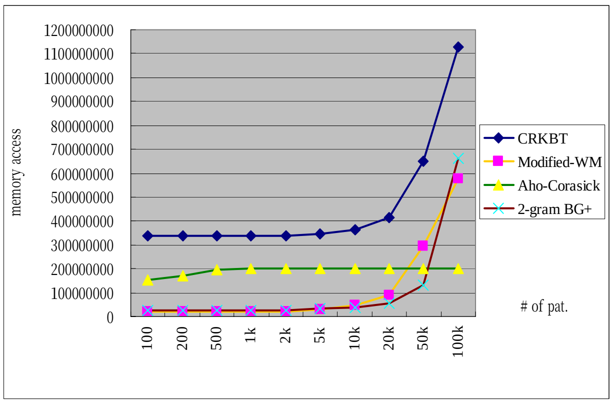

FIGURE 10 the number of memory accesses

The number of memory accesses from the program level is insufficient to justify the prior results. Because the penalty of L2 cache misses dominate the total performance. For this reason, we profile the number of L2 cache misses to verify the exceptional results.

Figure 11 shows the number of L2 cache misses for the CRKBT algorithm is less than that for the Modified-WM algorithm and the number of L2 cache misses for the 2-gram BG+ algorithm is the least. According with these results, we can easily prove the prior results, include that the CRKBT algorithm is more efficient than the Modified-MW algorithm as the pattern set size is more than 50,000. In addition, it is also proved again that the 2-gram BG+ algorithm has best efficiency under huge pattern sets.

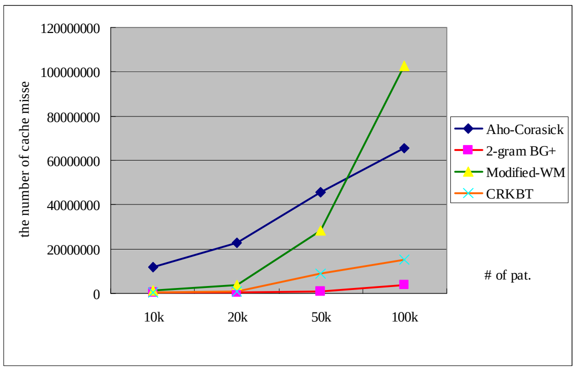

FIGURE 11 the number of L2 cache misses

The profiling results are summarized in figure 12.

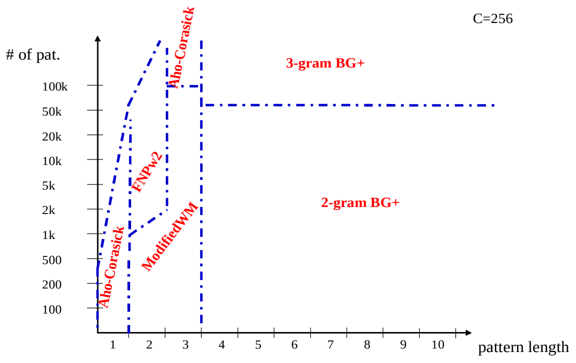

FIGURE 12 the profiling summary

Figure 13 concludes which package is located on which position and suits for which algorithm by means of the profiling results.

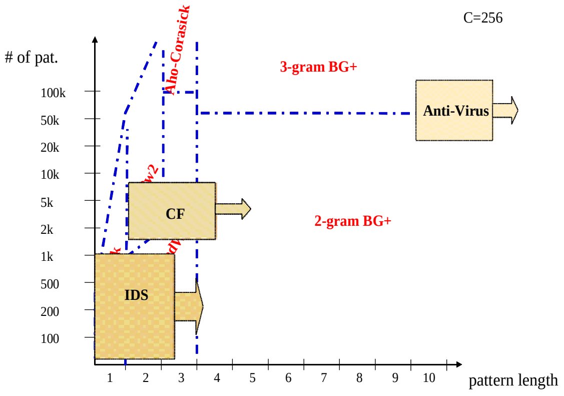

FIGURE 13 the profiling summary

IV. Conclusion

In this research, we present the CRKBT algorithm and prove it is more efficient than the RKBT algorithm in accordance with the external and internal benchmarking. The efficiency of the CRKBT algorithm is 4 times faster than the RKBT algorithm for huge pattern sets. Moreover, the BG+ and SOG+ algorithms that use it as the verification algorithm are also 2 times faster than the original algorithm.

The external and internal profiling shows the AC algorithm is suitable for LSP=1, the Modified-WM algorithm is suitable for the LSP between 2 and 3, and the 2-gram BG+ algorithm is suitable for LSP$\geq$4. These results are also justified by means of the real application experiments. Some applications have dramatically improvement as well. These results also help to select an efficient algorithm to design a novel application in the future.

In addition, this work also observes the difference between the real and synthetic data by means of the real application experiments. The anti-virus application is not sensitive to the synthetic data or the real data, because the character set distribution is close to uniform distribution. But the application of content filtering is sensitive to the data type, real data or synthetic data, because all patterns in the DansGuardian package is biased to English word. Moreover, we observe the bottleneck in content filtering application is to verify all potential matching in order to find out all matched content keywords, because the plenty of content keywords have the same hash value.

References

[1] S. Wu, and U. Manber, “A Fast Algorithm for Multi-pattern Searchin,” *Report TR-94-17, Department of Computer Science, University of Arizona*, 1994.

[2] J. Kytojoki, L. Salmela, and J. Tarhio, “Tuning String Matching for Huge Pattern Sets,” *CPM 2003*, LNCS 2676, pp. 211-224, 2003.

[3] A. Aho, and M. Corasick, “Efficient String Matching: An Aid to Bibliographic Search,” *Communications of the ACM 18,* pp. 333-340, 1975.

[4] Valgrind, http://valgrind.org/.