J.Natn.Sci.Foundation Sri Lanka 2018 46 (4): 547 - 557
DOI: http://dx.doi.org/10.4038/jnsfsr.v46i4.8630

RESEARCH ARTICLE

Average time complexity analysis of Commentz-Walter algorithm

S.D. Dewasurendra[1] and S.M. Vidanagamachchi[1,2*]
[1] *Department of Computer Engineering, Faculty of Engineering, University of Peradeniya, Peradeniya.*
[2] *Department of Computer Science, Faculty of Science, University of Ruhuna, Matara.*

Revised: 28 April 2018; Accepted: 25 May 2018

Abstract: The Commentz-Walter algorithm provides a suffix-based, exact multiple string matching procedure that combines the ideas of both Aho-Corasick and Boyer-Moore string matching algorithms. Aho-Corasick algorithm is known as the fastest exact multiple string matching algorithm with linear worst case time complexity, and Boyer-Moore algorithm utilises a shift based approach in handling mismatches in the matching phase to obtain a usually sub linear time complexity. Due to the complexity of the Commentz-Walter algorithm, a systematic and a comprehensive time complexity analysis has not yet been reported in the literature. In this article, a comprehensive average time complexity analysis of Commentz Walter algorithm is presented and it is used to formally explain empirical results reported elsewhere for exact and limited tolerance string matching. Performance analysis model permits 'complete' exploration of the solution space of Commentz-Walter with respect to all relevant parameters. The models developed here could be useful for other researchers to investigate the appropriateness of Commentz-Walter to specific application domains. The analysis method can be extended to cover Aho-Corasick and other string matching algorithms without much difficulty.

Keywords: Average time complexity, Commentz-Walter, expected shift length, multiple pattern matching.

INTRODUCTION

The primary objective of this article is to present a comprehensive average time complexity analysis of Commentz-Walter (CW) algorithm (Commentz-Walter, 1979), which has not been reported earlier. Aho-Corasick (AC) (Aho, 1975) and CW are two multiple pattern matching algorithms, where the AC has linear worst-case time complexity, O (n+k), while the CW is called a quadratic time algorithm of O (nm) complexity in the matching phase. Here n is the length of the input text to be searched, k is the number of pattern occurrences in the text and m is the length of the longest pattern. CW has been developed by embedding the functionality of Boyer-Moore algorithm (shifts) in AC algorithm. AC and CW are developed on graph and tree structures, respectively, and both interpreted as finite state machines (FSMs) with the remaining text to be analysed at each alignment as the input tape. The data structures are built either with suffixes or prefixes of the key words: starting with an initial state 0, AC constructs a trie on prefixes of the keyword set, whereas CW does it on the suffixes of reversed keywords. In addition to the trie construction in the pre-processing stage, AC requires constructing failure links that may be needed whenever a mismatch occurs during the searching phase. CW does not construct failure links, instead, it uses three shift values [eg: shift1, shift2, and char (a)] calculated in the pre-processing stage, to calculate a total shift value in run time whenever a mismatch occurs. Both algorithms achieve economy of search at each alignment by searching along the current branch of the tree ignoring all the other branches of the tree. In the pre-processing stage, the CW calculates three shift values considering proper suffixes of keywords (shift1), suffixes with output patterns (shift2) and next character occurrences (char (.)) of the keyword sets. Figure 1(a) and (b) show the difference between the two FSMs for a set of patterns he, her, his, she and hers.

* Corresponding author (smv@dcs.ruh.ac.lk; https://orcid.org/0000-0002-2245-4527)

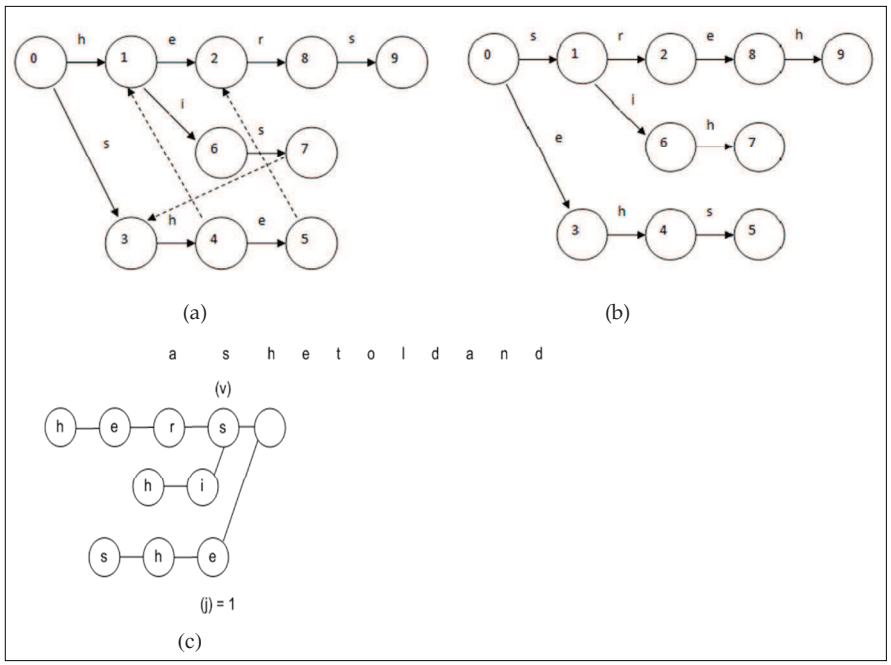

Figure 1: (a) Prefix based FSM implementation of AC algorithm for a keyword set of he, she, hers and his (Aho, 1975); (b) suffix based FSM implementation of CW algorithm (Vidanagamachchi *et al*., 2012); (c) a search instance of CW.

In the original work of CW, two algorithms were introduced (algorithm B and B1) combining the concepts of AC and Boyer-Moore (BM) algorithms (Boyer & Moore, 1977) and they have the presented time complexities of algorithm B (Commentz-Walter, 1979). Based on the findings, it is reported that the searching phase of algorithm B performs sub-linear on the average and it has a non-linear worst case time complexity (Commentz-Walter, 1979). To that end, aiming to overcome the problems in algorithm B, they have proposed algorithm B1, which is known to be having linear worst case time complexity (Commentz-Walter, 1979). Although algorithm B1 seems to perform better, it was recommended to use algorithm B in practical situations as algorithm B1 is having a high overhead (Commentz-Walter, 1979). A lower bound for the average number of characters examined in a search for random patterns in a text over a uniformly distributed alphabet has been provided in Navarro *et al*. (2004). In Barton and Iliopoulos (2014), two of the algorithms for pattern matching, where wildcards are restricted to either the pattern or the text are analysed for average time complexity, and lower bounds provided. However, these results do not help in doing a proper exploration of the *problem* space for finding problem configurations better solved through CW or any particular string matching algorithm. Of particular interest is the work in Kouzinopoulos & Margaritis (2011) where the performance of the pre-processing phase is evaluated in terms of running time of the presented multiple keyword matching algorithms, CW and Wu-Munbar. This work can complement the present study on running time.

Analysis of a suitable algorithm for Snort (open-source, free and lightweight network intrusion detection system) was carried out by James (2006) and it is mentioned that they have not implemented algorithm B1 due to its daunting complexity (theoretically and empirically). Further there were no implementations found on algorithm B1 to the best of their knowledge (James, 2006). The next paragraph describes the past work on the empirical performance of the AC and CW [based on algorithm B (Commentz-Walter, 1979)] algorithms.

Although Shoshana (2009) reports the CW algorithm to be generally faster than the AC algorithm, it does not report any analysis or limit of the keyword size and numbers of keywords. In Van (1990), it is reported that with small numbers of keywords, the BM aspect of CW can make it faster than AC algorithm. However, based on the findings by Watson, the actual performance of CW gets decreased with increasing keyword sizes with more than 13 keywords (Watson, 1994; 1995). As reported in

Watson (1994), CW is better than AC with long patterns (20 – 120), although there is no critical analysis with large pattern sets on that. Given the lack of general agreement on the performance of CW and the key role played by the shift operator and its heavy dependence on domain specific characteristics of the keys and the input texts, there is still much to be desired in terms of formal explanation for the average time behaviour of CW.

Against this background we took a closer look at the computational performance of the algorithm B of CW in the particular context of protein identification, a problem we characterise in Vidanagamachchi (2017). Although it is assumed that algorithm B of CW is practically applicable in many applications (Commentz-Walter, 1979), no complete implementation (freely available) was found for algorithm B of CW though one was reported in Watson (1994). We started by implementing the algorithm B of CW to compare its empirical performance with AC in peptide identification applications and the results were presented in our previous work (Vidanagamachchi *et al*., 2012). Next we completed a detailed analysis of the average time complexity of CW applicable to the peculiarities of proteomics, and easily extendable to other problem domains with specific parameter profiles. The following section describes details of average time complexity analysis of CW algorithm, which would now permit a formal interpretation of different performance results reported in literature.

The remaining sections of this paper describe the performance analysis of these two algorithms in detail. The details of average complexity analysis (methods) are given in the next section. Analysis of our results with previously reported empirical results of CW follows. Further it discusses the average time complexity of limited tolerance version of CW algorithm.

METHODOLOGY

Through a careful examination of CW algorithm (B), it was recognised (as has been observed by earlier researchers too) that the key issue that determines the average running time of the algorithm is the interplay among the three computations: online component of the shift length computation, shift length and search time within an alignment before a shift computation becomes necessary.

We tried to discover any order of magnitude relation between the online and offline components of computation in the unstructured general case of random multiple substring matching in unstructured input texts of varying lengths, solely based on the algorithmic structure, the offline components being,

(i) building the reversed trie and
(ii) computation of functions,
out: out(.), depth: d(.), word: w(.), set1(.), set2(.), W_{min}(.), shift1(.), shift2(.) and char(.);

and the online components being,

(i) search within the reversed trie for a given alignment before a shift computation becomes necessary,
(ii) shift length computation using stored offline computational results and the results of (i), and
(iii) shifting the trie and getting a new alignment with the input text.

What we discovered was that there was no such general relation and that the relative magnitudes are heavily context dependent, meaning that there was a possibility of discovering relationships where problems had some structure.

This observation was substantiated by the varied outcomes in empirical studies reported in the literature.

Hence we set forth discovering a method of representing the structure of a given problem context which would permit a meaningful exploration of the *problem* space (note: not the *solution* space).

This led to the notion of a *generic trie* for a particular problem context.

In our case we observed that the main concern is online computations and hence the average times were computed for those. However, there may be problem contexts where this may not be the case, and our methodology is valid for those cases too.

We next formulated a set of expressions that permitted us to explore the complete parameter space for any particular problem context to compute average computational times and validated the methodology through its application to CW algorithm.

This methodology is directly applicable to the systematic analysis of any other string matching algorithm.

As far as the general procedure followed in developing the set of expressions is concerned, it is based on the parameterised computation of probabilistic expectations of the average computational times as follows, and explained in detail in the next section.

Let A be an algorithm, S_n be the sample space of all inputs of size n, $T_n(i)$ the number of steps (comparisons) taken by A on input i in S_n and $P_r(i)$ the probability of input i based on the probability distribution over S_n. Then, the average running time of A given S_n is $E(T_n)$. The average/ expected time complexity $E(T_n)$ is given by equation (1).

$$E(T_n) = \sum_{i \in S_n} P_r(i) T_n(i) \quad ...(1)$$

RESULTS AND DISCUSSION

Let P_m be the probability of matching a character in the search phase of CW, b_T the branching factor of the reversed keyword trie and j the distance from the root (level) to the current character, v, being considered in the trie [Figure 1(c)].

Then, the expected number of character comparisons made in the first alignment of the reversed trie with input text before a match is found when $j=1$ can be obtained as in equation (2) below.

$$E_{11}^S = P_m + 2(1- P_m) P_m + 3(1- P_m)^2 P_m + \ldots + b_T(1- P_m)^{b_T - 1} P_m$$

$$= P_m + \sum_{k=2}^{b_T} k(1 - P_m)^{(k-1)} P_m \quad ...(2)$$

Then, the number of comparisons made in the first alignment before a shift is made when $j = 1$ can be obtained as,

$$E_{1S1}^S = b_T \quad ...(2')$$

The probability that there will be a shift in the first alignment when $j=1$ is $(1- P_m)^{b_T}$. A shift here is as defined in Commentz-Walter (1979), where the root of the reversed trie is shifted to the right along the input text string by an amount computed by the CW algorithm.

Then, the probability that there will be a shift in the first alignment when $j \leq 1$ is

$$P_{j1}^{sa1} = (1- P_m)^{b_T}$$

Let $\alpha = (1- P_m)^{b_T}$

$$P_{j1}^{sa1} = \alpha \quad ...(3)$$

Therefore, the probability that there will not be a shift in the first alignment when $j = 1$ is $1-\alpha$.

Then, assuming the branching factor to remain as b_T at each node of the reversed trie, the expected number of comparisons in the first alignment before a match is found, when $j = 2$ can be given as in equations (4) and (5).

$$E_{12} = E_{11}^S (1-P_{j1}^{sa1}) + E_{12}^S$$

$$= E_{11}^S (1-\alpha) + E_{12}^S \quad ...(4)$$

$$E_{11}^S = E_{12}^S \text{ (by assumption that } b_T \text{ remains the same)} \quad ...(5)$$

The assumption in equation (5) is justified by the isomorphism between sub trees of the trie in the search. We will generalise this assumption to include $E_{11}^S = E_{ij}^S, \ \forall i, j$.

Then, the expected number of comparisons made in the first alignment before a shift is made when $j = 2$ can be obtained as,

$$E_{1S2}^S = E_{11}^S (1-\alpha) + b_T \quad ...(4')$$

Then, the probability that there will be a shift in the first alignment when $j \leq 2$ will be as in equation (6)

$$P_{j2}^{sa1} = (1- P_m)^{b_T} + [1 - (1- P_m)^{b_T}] (1- P_m)^{b_T}$$

$$= \alpha + (1-\alpha)\alpha$$

Let $\beta_1 = \alpha + (1-\alpha)\alpha$

Hence, $P_{j2}^{sa1} = \beta_1 \quad ...(6)$

Similarly we can obtain the probability that there will be a shift in the first alignment when $j \leq 3$ is as P_{j3}^{sa1} = Probability that there was a shift when $j \leq 2$ + (Probability that there was no shift when $j \leq 2$) × (Probability that there was a shift when $j = 2$) follows:

$$P_{j3}^{sa1} = \beta_1 + (1-\beta_1)\alpha$$

Let $\beta_2 = \beta_1 + (1-\beta_1)\alpha$

Which gives

$$P_{j3}^{sa1} = \beta_2 \quad ...(7)$$

Likewise we obtain

$$P_{j4}^{sa1} = \beta_3 \quad = \beta_2 + (1-\beta_2)\alpha \quad ...(8)$$

Generalising the above we get equation (9) as the probability that there will be a shift in the first alignment

when $j \leq w_{Min}$, where w_{Min} is the length of the shortest pattern.

$$P_{jw_{Min}}^{sa1} = \beta\, w_{Min\,-1} \quad = \beta\, w_{Min\,-2} + (1-\beta\, w_{Min\,-2})\alpha \quad ...(9)$$

Then the probability that there will not be a shift in the first alignment is,

$$1 - P_{jw_{Min}}^{sa1} = 1 - \beta\, w_{Min\,-1} \quad ...(10)$$

Then, expected value of *'j'* at which a shift will happen in the first alignment can be obtained as in equation (11).

$$E_1^s(j) = 1\times\alpha + 2\times(1-\alpha)\alpha + 3\times(1-\beta_1)\alpha + 4\times(1-\beta_2)\alpha + \ldots + w_{Min}\times(1-\beta_{w_{Min}-2})\alpha$$

$$= \alpha[3 - 2\alpha + \textstyle\sum_{k=3}^{w_{Min}} k(1-\beta_{k-2})] \quad ...(11)$$

Continuing from (2, 3, 4, 5 and 6), the expected number of comparisons in the first alignment before a match is found when $j = 3$ is

$$E_{13} = E_{12}\,(1-P_{j2}^{sa1}) + E_{13}^S$$

$$= (E_{11}^S\,(1-\alpha) + E_{12}^S)\,(1-P_{j2}^{sa1}) + E_{13}^S$$

$$= (E_{11}^S\,(1-\alpha) + E_{11}^S)\,(1-\beta_1) + E_{11}^S$$

$$= [(2-\alpha)(1-\beta_1)+1]\,E_{11}^S \quad ...(12)$$

Then, the expected number of comparisons in the first alignment before a match is found when $j = 4$ is

$$E_{14} = E_{13}\,(1-P_{j3}^{sa1}) + E_{14}^S$$

$$= [(E_{11}^S\,(1-\alpha) + E_{11}^S)\,(1-\beta_1) + E_{11}^S]\,(1-\beta_2) + E_{11}^S$$

$$= [(2-\alpha)(1-\beta_1)+1]\,E_{11}^S\,(1-\beta_2) + E_{11}^S$$

$$= \{[(2-\alpha)(1-\beta_1)+1]\,(1-\beta_2)+1\}\,E_{11}^S$$

$$= [(2-\alpha)(1-\beta_1)(1-\beta_2) + (1-\beta_2) + 1]\,E_{11}^S$$

$$E_{15} = [(2-\alpha)(1-\beta_1)(1-\beta_2)\,(1-\beta_3) + (1-\beta_2)(1-\beta_3) + (1-\beta_3) + 1]\,E_{11}^S$$

Generalising which, we obtain,

$$E_{1j} = [(2-\alpha)(1-\beta_1)(1-\beta_2)(1-\beta_3)\ldots\,(1-\beta_{j-2}) + (1-\beta_2)(1-\beta_3)\ldots\,(1-\beta_{j-2}) + \ldots + (1-\beta_{j-2}) + 1]\,E_{11}^S$$

Hence,

$$E_{1j} = \{[\textstyle\prod_{k=1}^{j-2}(1-\beta_k)]\,(2-\alpha) + \sum_{l=1}^{j-2}[\prod_{k=l+1}^{j-2}(1-\beta_k)] + 1\}\,E_{11}^S \quad ...(13)$$

Then, the expected number of comparisons in the first alignment before a match is found when $j = E_1^s(j)$, can be written as in equation (14) assuming $E_{ij}^S = E_{11}^S, \forall i, j$.

$$E_{1E_1^s(j)} = \{[\textstyle\prod_{k=1}^{E_1^s(j)-2}(1-\beta_k)]\,(2-\alpha) + \sum_{k=l+1}^{E_1^s(j)-2}[\prod_{k=l+1}^{E_1^s(j)-2}(1-\beta_k)] + 1\}\,E_{11}^S \quad ...(14)$$

Then, the number of comparisons made in the first alignment before a shift is made when $j = (j)$ can be obtained as,

$$E_{1SE_1^s(j)}^S = \{[\textstyle\prod_{k=1}^{E_1^s(j)-2}(1-\beta_k)]\,(2-\alpha) + \sum_{k=l+1}^{E_1^s(j)-2}\prod_{k=l+1}^{E_1^s(j)-2}(1-\beta_k)\}\,E_{11}^S + b_T \quad ...(14')$$

Now, to find the length of the shift, it is necessary to know the node, v that led to the shift (the parent node of the set of nodes that failed to produce a match). Since there is no way that this can be guessed we take the following approach to get an average estimate for the same. The maximum possible shift when there is a mismatch is w_{Min}. We can safely assume that the average shift length is $(1+shiftFactor \times (W_{Min} - 1))$ each time a shift happens (as the minimum shift is 1) for some $0 \leq shiftFactor \leq 1$.

If the text string is very much longer than the longest key (= the depth of the trie), we can neglect the edge effects and consider only the alignments where the trie falls completely within the input text. Then, extending equation (11), the expected value of *'j'* at which a shift will happen in the i^{th} alignment can be defined as in equation (15).

$$E_i^s(j) = \alpha[3 - 2\alpha + \textstyle\sum_{k=3}^{d_T} k(1-\beta_{k-2})] \quad ...(15)$$

Here d_T is the depth of the reversed trie, being equivalent to $max\ \{m(0), \ldots, m(p-1)\}$, the maximum keyword length, where $m[0\ldots(p-1)]$ is the array of keyword lengths.

Then, the expected number of comparisons in the alignment *'i'* before a match is found when $j = E_i^s(j)$ is $E_{iE_i^s(j)}$ as shown in equation (16).

$$E_{iE_i^S(j)} = \{[\prod_{k=1}^{E_i^S(j)-2}(1-\beta_k)](2-\alpha) + \sum_{l=1}^{E_i^S(j)-2}[\prod_{k=l+1}^{E_i^S(j)-2}(1-\beta_k)] + 1\}E_{11}^S \quad ...(16)$$

Then, the number of comparisons made in the alignment *'i'* before a shift is made when $j = E_i^S(j)$ can be obtained as,

$$E_{iSE_i^S(j)}^S = \{[\prod_{k=1}^{E_i^S(j)-2}(1-\beta_k)](2-\alpha) + \sum_{l=1}^{E_i^S(j)-2}\prod_{k=l+1}^{E_i^S(j)-2}(1-\beta_k)\}E_{11}^S + b_T \quad ...(16')$$

Extending equation (9) for the i^{th} alignment, the probability that there will be a shift in the i^{th} alignment when $j \leq d_T$ where d_T is the depth of the reversed trie is obtained as,

$$P_{jd_T}^{sai} = \beta_{d_T-1} = \beta_{d_T-2} + (1-\beta_{d_T-2})\alpha \quad ...(17)$$

which is also the probability that there will be a shift in the i^{th} alignment,$(P(s)_i)$ (18).

$$P(s)_i = P_{jd_T}^{sai} = \beta_{d_T-1} = \beta_{d_T-2} + (1-\beta_{d_T-2})\alpha \quad ...(18)$$

Then, under the assumption above that the average shift length is $(1+shiftFactor\times(W_{Min}-1))$ each time a shift happens, the expected shift length in the i^{th} alignment $(E(s)_i)$ can be computed as in equation (19), which also happens to be the general expression for the expected shift length in an alignment when edge effects are neglected as assumed above.

$$E(s)_i = (1+shiftFactor\times(W_{Min}-1))\times[\beta_{d_T-2} + (1-\beta_{d_T-2})\alpha] \quad ...(19)$$

where, $0 \leq shiftFactor \leq 1$.

Therefore, the expected number of alignments in the matching process is

$$n/E(S)_i \quad ...(20)$$

The n above is the length of the input text. The expected number of comparisons made in the matching process can then be obtained as

$$E(c) = [n/E(S)_i]\times E_{iSE_i^S(j)}^S \quad ...(21)$$

According to the above analysis, the expected number of comparisons (average time complexity) depends on the expected shift length. Next, a comparison of the derived average time complexity of CW with previously published results was carried out.

The whole idea is to systematically explore the solution space of CW algorithm applications. Except for empirical results and general conjectures, no formal model of the solution space has been reported. The model we propose in this paper permits conducting such exploration by feeding in application specific parameters. The model helps in formally establishing the empirical results reported elsewhere and to investigate the performance of CW for particular applications. In this process of modelling, certain of these parameters get only implicit representation: e.g., the number of keywords, '*p*', the lengths of individual keywords, *m*[*0 ... (p-1)*], the keyword array, *x*[*0 ... (p-1)*], and the input text (byte) array, *y*[*1 ... n*]. This is because they are too application specific to be captured in any general modelling framework. Their representation is indirect, captured through the specification of b_T, d_T, W_{min}, n, *shiftFactor* and P_m. To demonstrate how general topologies of input tries can be generated to represent typical keyword length distributions, the following tries may be used for the case where the keyword lengths vary between W_{min} and d_T in a uniform manner: each line segment represents one character from the input alphabet (Figure 2). Figure 2(a) would stand for $p = 1/2 \times (d_T - W_{min} + 1)\times(W_{min} + d_T)$. A possible way to use these generic forms in a performance evaluation would be to select the topology that could best describe the keyword set in question, and use that topology to get an estimate of p as indicated in Figure 2(a).

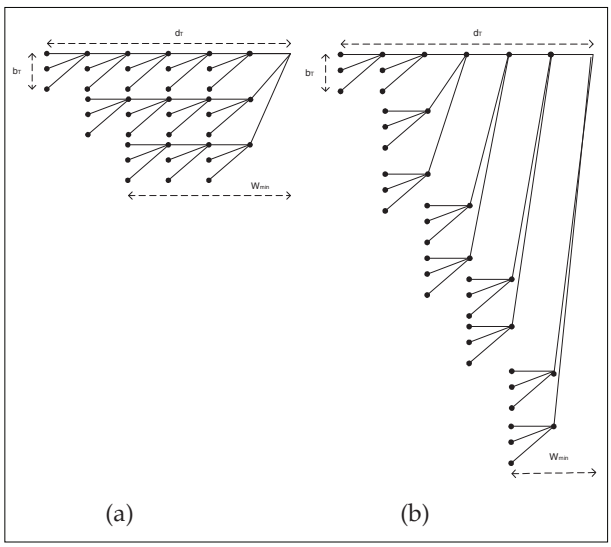

Figure 2: Two generic tries with word lengths uniformly varying between W_{min} and d_T

As stated in the original paper on CW algorithm (Commentz-Walter, 1979), total running time in matching (including scanning and shifts) depends on the total number of character comparisons. According to the results reported by Commentz-Walter (1979), the average number of references to a document/text character is low when the length of the keywords is high. For example for a keyword of length 3, the number of average character references is around 0.85 and when the length grows up to 11, it decreases to 0.4 (approximately) for 2 keywords. This behaviour can be explained with the length of the smallest pattern (w_{Min}) in our average time complexity analysis. If w_{Min} is high, the number of character references in the text is likely to decrease as the shift length can increase up to w_{Min}. This is supported by our analysis: according to equation (19), the expected shift length becomes high whenever the w_{Min} is high. Therefore, the expected number of alignments based on equation (20) becomes low and the expected number of references becomes low according to equation (21).

Further, for long keywords, test results presented in Commentz-Walter (1979) indicate that the average number of references to a document/text character increases slightly with higher numbers of keywords. For example, for keywords of length 11, the average number

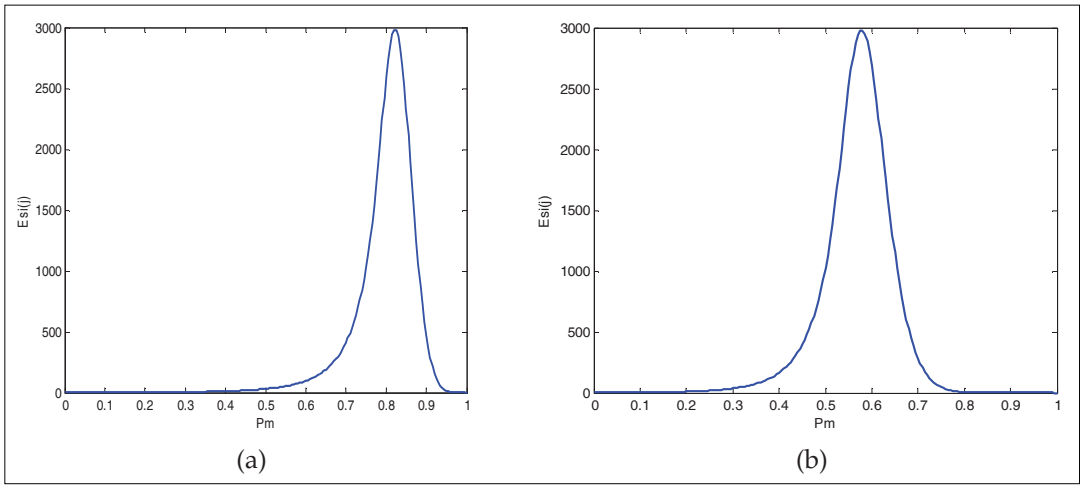

Figure 3: The expected value of '*j*' at which a shift will happen in the i^{th} alignment for d_T = 10,000 (a)b_T = 5; (b) b_T = 10

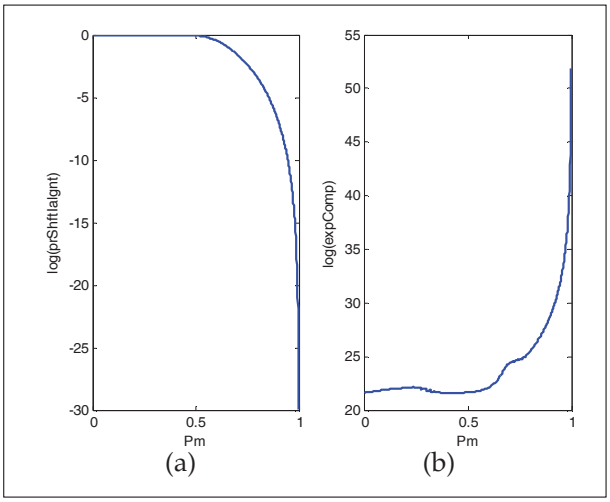

Figure 4: Variation of (a) $P(s)_i$ and (b) $E(c)$ with P_m, for b_T = 5, d_T = 100, w_{Min} = 50, length of input text = 10000000000, *shift-factor* = 0.4

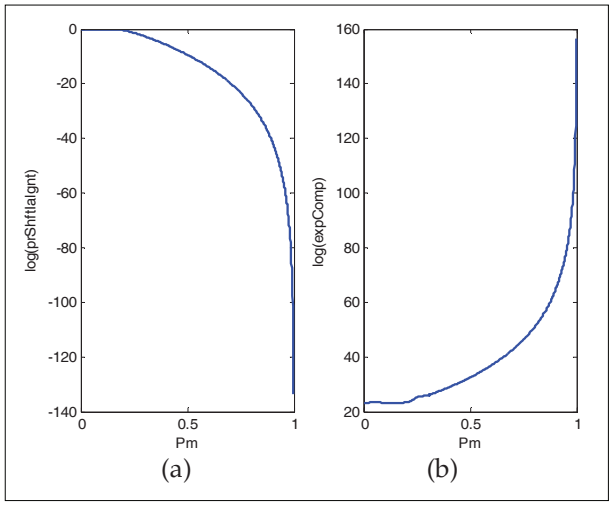

Figure 5: Variation of (a) $P(s)_i$ and (b) $E(c)$ with P_m, for b_T = 20, d_T = 100, w_{Min} = 50, length of input text = 10000000000, *shift-factor* = 0.4

of character references are 0.4 for a keywords set of 2 while the number of character references increased up to 0.8 for keywords set of 64. In our own empirical studies conducted earlier, we also observed similar effects for the comparable keyword configurations (Vidanagamachchi *et al*., 2012), which we could attribute to characteristics of the keyword set used in the present experiment. However, since these results, both in Commentz-Walter (1979) and Vidanagamachchi *et al*. (2012), relate to very specific experimental configurations, drawing general conclusions based on them is questionable.

According to the present analysis, b_T increases when there are more keywords presented in the trie and also with the number of characters in the alphabet, n_a: theoretically the maximum possible branching factor is $min\{p, n_a\}$, where p is the number of keywords. Further, probability of matching a character, P_m (which is the probability that any given character reference is a success), may also decrease with p. The effect on the value of $(1- P_m)^{b_T}$ would depend on interplay between the increase of b_T and the decrease of P_m. If the latter effect dominated, then it would cause the expected shift length in equation (19) to increase, thereby decreasing the number of references to the document characters; otherwise, the reverse would happen. This phenomenon is clearly visible in Figures 4 and 5 obtained by simulating our expressions in equations (18) and (21).

Note that the abscissa in Figures 4 and 5 are log-scales. Figure 3 indicates how increasing b_T could increase expected 'j' at which a shift could happen in a given alignment: when b_T increases, 'j' increases, thus increasing the number of character comparisons. In fact the same increase is seen for short keywords, for instance, of length 2 to 5 (Figures 6 and 7).

Again, according to Commentz-Walter (1979), average number of references to a document character is lower when the number of words in the keyword set and length of the keywords are higher. However, this is clearly not the case according to our analysis as easily seen by regenerating any one of Figures 4 to 7 with corresponding higher values for b_T, d_T and W_{min} using equation (21). On the other hand when W_{min} is increased while keeping the other parameters the same, there is an immediate reduction in the average number of character references, as explained earlier, which is intuitively correct.

According to Watson (1994), Jony (2014) and Aho (1990), with small numbers of keywords, the Boyer-Moore aspects of the CW algorithm can make it faster than the Aho-Corasick (AC) algorithm, but with larger

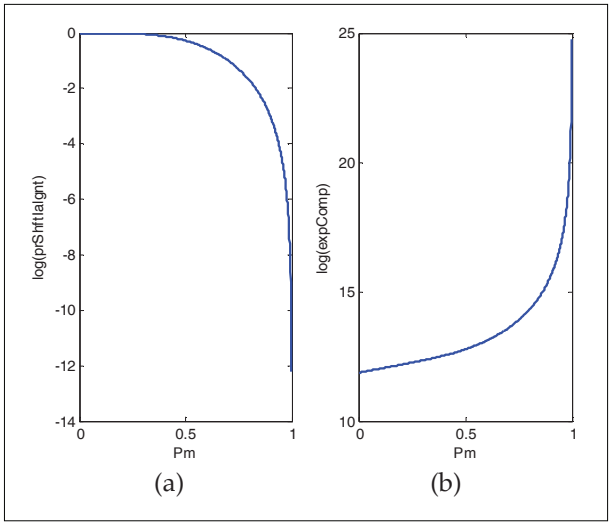

Figure 6: Variation of (a) $P(s)_i$ and (b) $E(c)$ with P_m, for $b_T = 2$, $d_T = 5$, $w_{Min} = 2$, length of input text =10000, *shift-factor* = 0.4

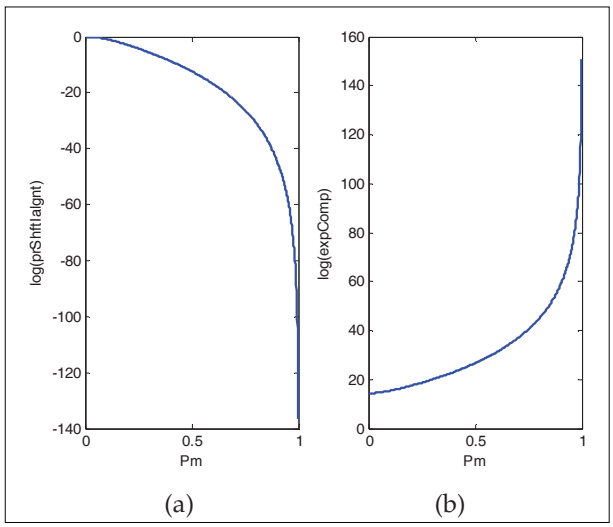

Figure 7: Variation of (a) $P(s)_i$ and (b) $E(c)$ with P_m, for $b_T = 20$, $d_T = 5$, $w_{Min} = 2$, length of input text =10000, *shift-factor* = 0.4

numbers of keywords, the AC algorithm has a slight edge. This can be investigated in our model using varying numbers of keywords to compute the expected number of comparisons in an alignment 'i' before a shift is made. A simulation of equation (16') with the estimator $p = 1/2 \times (d_T - W_{min} + 1) \times (W_{min} + d_T)$ given for the generic trie in Figure 2(a) gives the value for $E^S_{iSE^s_i(j)}$ as 2.33 for $p = 28$, $P_m = 0.1$, $b_T = 2$, $d_T = 5$ *and* $W_{min} = 2$; whereas $E^S_{iSE^s_i(j)}$ comes out as 15.00 for $p = 140$, $P_m = 0.1$, $b_T = 10$, $d_T = 5$ and $W_{min} = 2$.

This type of numerical experimentation with our model can provide stronger justification for claims based solely on empirical studies. Our own empirical results related to this were reported in Vidanagamachchi *et al*. (2012), where the matching time of AC algorithm as well as CW was seen to increase with increasing numbers of keywords. Work reported in Jony (2014) for text matching claims that, when the number of keywords is held constant, running time of CW algorithm gets increased gradually with increasing lengths of keywords. In contrast, according to the analysis of the original paper (Commentz-Walter, 1979), the number of character references to the text decreased with the increasing length of patterns. It was claimed that more shifts were required with longer keywords, and hence, running time should get decreased if the shifts were large enough (this is bounded by w_{Min}).

On the other hand, the I/O required for handling a mismatch may increase the time of matching as well, as we reported in Vidanagamachchi *et al*. (2012). According to the current analysis, when the length of the patterns becomes high, w_{min} and d_T should also become high. Therefore, the expected number of comparisons [E(c)] in equation (21) gets decreased as $E(s)_i$ in equation (19) increases. However, when the pattern length increases, there is a high possibility of increasing the expected number of comparisons in the alignment 'i' before a shift is made ($E^S_{iSE^s_i(j)}$), as can be seen by simulating equation (16'), which could cause [E(c)] to get increased, thus increasing the average running time. Here the interplay between the ratio of n/ $E(S)_i$ and $E^S_{iSE^s_i(j)}$ can be seen to determine the effect increasing keyword lengths. If the former decreased gradually while $E^S_{iSE^s_i(j)}$ increased, balancing the effects, then the above claims could lose their strength, as observed in Vidanagamachchi *et al*. (2012).

The work of Pandiselvam *et al*. (2014) and James (2006) do not include any proteomics or other text analysis regarding CW time complexity except comparisons of string matching algorithms. Therefore their results are not discussed.

In order to put the work reported in this paper in proper perspective, consider the case of Commentz-Walter (1979), where the keywords were 100 titles of English and German books on Computer Science, which were strictly exact character sequences, and the input text was an exact string of titles, using an alphabet of 37 alphanumeric characters. Their analysis was based on the average number of character references they observed choosing length of the keywords from 3, 5, 7, 9 and 11 and number of keywords from 2, 4, 8, 16, 32 and 64. We can compare this with proteomic string matching with an alphabet of 20 characters, representing the known set of amino acids and much longer peptides with isoforms acting as keywords and very much longer peptide sequences with isoforms and variants acting as input texts: inexact keywords to be matched against inexact input strings. P_m may decrease with n_a, minimum possible value in a randomised experiment being $1/n_a$, with even *0* possible, otherwise.

The characterisation we did on proteomic string matching reveals a high degree of variants and isoforms

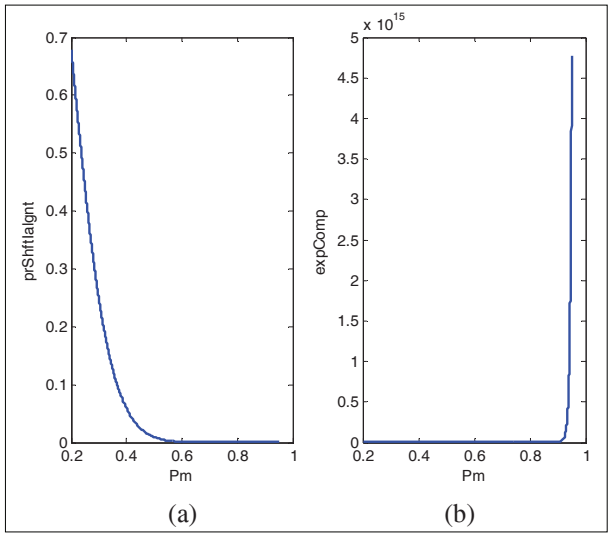

Figure 8: Variation of (a)$P(s)_i$ and (b)$E(c)$ with pMatch, for $b_T = 10$, $d_T = 10$, $w_{Min} = 5$, lenInTxt =1000

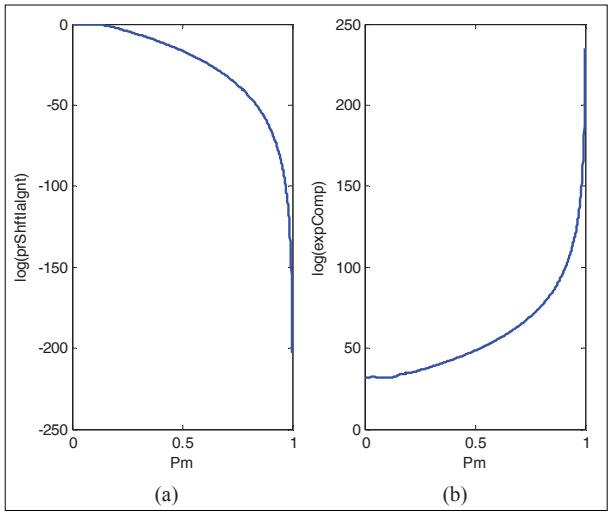

Figure 9: Variation of (a) $log(P(s)_i)$ and (b) $log(E(c))$ with pMatch, for $b_T = 30$, $d_T = 79$, $w_{Min} = 10$, lenInTxt =110^{13}

of both peptides and proteins, which prompted us to pose this problem as a limited tolerance inexact string matching problem (Vidanagamachchi, 2017). The next section briefly looks at how the formulas obtained above can be used in this latter context.

Average time complexity of the limited tolerance version

The limited tolerance algorithm was designed to solve the problem of protein identification with isoforms and post-translational modifications by modifying the implementation of Aho-Corasick algorithm. This limited tolerance idea can easily be extended to CW algorithm. The finite state machine of AC algorithm was developed using peptide patterns generated from protein digestion. Then the input text is a protein assembled from digested peptides (eg: unreviewed protein in a database or a set of assembled peptides from a sample). For example, consider the peptides digested (in-silico) from a generated reference protein:

Reference protein: 'MMDA (A or R or D) CRDDCAM (M or C)'
Peptides generated: 'MMDA (A or R or D) CR' and 'DDCAM (M or C)'

Then, Aho-Corasick FSM can be generated as follows; Figure 10(a) shows how the tolerance can be handled using the original AC algorithm. We modified this as shown in Figure 10(b) for the proposed limited tolerance variant.

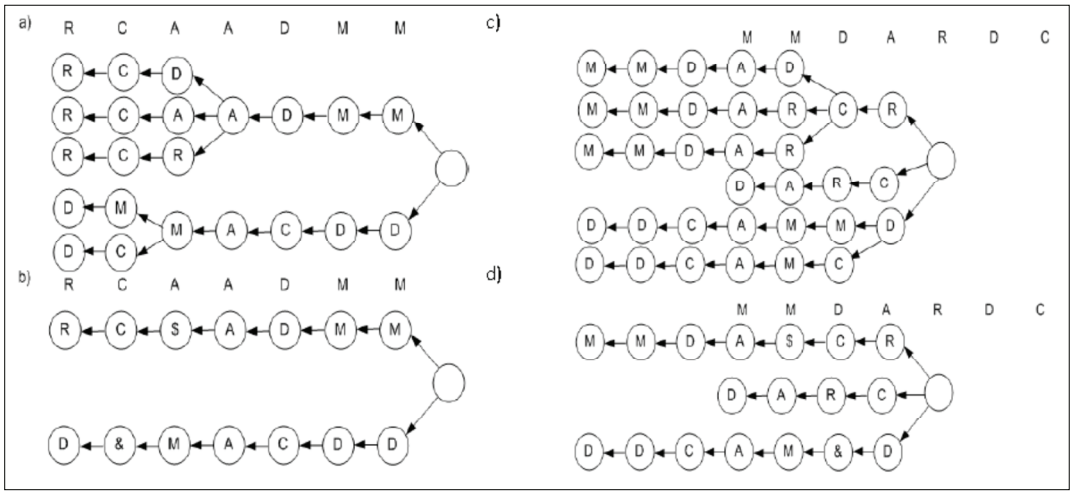

Figure 10: (a) Handling tolerance using original AC algorithm; (b) AC algorithm with limited tolerance; (c) handling tolerance using original CW algorithm; (d) CW algorithm with limited tolerance

For example, if peptides digested from the generated reference protein 'MMDA (A or R or D) CRDDCAM (M or C)' are 'MMDA (A or R or D) CR', 'DDCAM (M or C)' and DARC, then Commentz-Walter FSM can be generated as follows; Here the minimum length of a pattern is 4 and the position 5 of the input text (protein sequence) was aligned to the root of the FSM. Figure 10(c) shows how the tolerance can be handled by the original CW algorithm. We modified it as Figure 10(d) for the limited tolerance variant.

According to the time complexities of the analysis, it is shown that the limited tolerance methodology can save computational time by reducing the branches. According to the original versions of AC and CW, the tree may become larger with all possible tolerances; the time for searching should change with them too. The value of b_T varies based on the allowed tolerances and its value becomes high if the number of variants allowed per position is high. Therefore, it will affect the results of the entire analysis [equations from (2) to (20) in the complexity analysis] when applying them for peptide identification with limited tolerance in both AC and CW algorithms. As we have demonstrated through our experiments (Vidanagamachchi, 2017), the limited tolerance variant of AC results in significant performance improvement. It is also very likely to have a considerable effect on average time complexity of CW algorithm, when used in proteomic applications. The analysis provided in this paper can easily be used for this variant of CW and AC by pre-processing the keyword tries as indicated in Figure 10.

CONCLUSION

The average complexity analysis of CW algorithm provides a comprehensive and a systematic analysis (theoretically), which was not available in literature. It helps to formally interpret previously reported empirical results, as well as to do a systematic exploration of the solution space of CW. We have provided implements to capture parameters that cannot be explicitly incorporated as inputs to the generic performance evaluation models, thus permitting a systematic evaluation of their effects too. The importance of this nature of performance evaluation model for CW is highlighted in the context of heavy application dependence of runtime performance. The set of equations can be used easily by other researchers to carryout comprehensive investigation of average time performance in specific application areas.

The application to limited tolerance inexact multiple string matching context is also new and can save computing time by exploiting mutations/variations present within peptide sequences to reduce the branching factor, b_T, and keyword lengths in the keyword trie.

More specific to the numerical results obtained by our simulation of equations (2) to (21), we observed that the expected shift length (this is always less than the length of the smallest keyword, w_{Min}) and the branching factor, b_T, which is directly related to the number of characters in the input alphabet, have the greatest impact on average time performance of CW algorithm. The effect of keyword length distribution and the number of keywords can be analysed based on generic trie structures that can be built to represent particular application domains following the guidelines provided in our work.

Acknowledgement

The authors convey their sincere gratitude to Dr R.G. Ragel and Prof. M. Niranjan who are partners in the main project group where this research work was carried out, for early work with them led the authors to undertake the current investigation.

REFERENCES

Aho A.V. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM* **18**(6): 333 – 340. DOI: https://doi.org/10.1145/360825.360855

Aho A.V. (1990). Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science*, volume A, pp. 255 – 300. MIT Press, Cambridge, MA, USA. DOI: https://doi.org/10.1016/B978-0-444-88071-0.50010-2

Barton C. & Iliopoulos C.S. (2014). On the average-case complexity of pattern matching with wildcards. *Leibniz International Proceedings in Informatics*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany. arXiv:1407.0950 [cs.DS]

Boyer R.S. & Moore J.S. (1977). A fast string searching algorithm. *Communications of the ACM* **20**(10): 762 – 772. DOI: https://doi.org/10.1145/359842.359859

Commentz-Walter B. (1979). A string matching algorithm fast on the average. *Proceedings of the 6th International Colloquium on Automata, Languages and Programming*, volume 71, pp. 118 – 132. Springer, Berlin, Germany. DOI: https://doi.org/10.1007/3-540-09510-1_10

James K. (2006). An examination of pattern matching algorithms for intrusion detection systems. *Masters thesis*, Ottawa-Carleton Institute for Computer Science, Canada.

Jony A.I. (2014). Analysis of multiple string pattern matching algorithms. *International Journal of Advanced Computer Science and Information Technology* **03**(04): 344 – 353.

Kouzinopoulos C.S. & Margaritis K.G. (2011). A performance evaluation of the pre-processing phase of multiple keyword matching algorithms. *Proceedings of the 15th Panhellenic Conference on Informatics (PCI)*, Kastoria, Greece, 30 September – 2 October, pp. 85 – 89.

Navarro G. & Fredriksson K. (2004). Average complexity of exact and approximate multiple string matching. *Theoretical Computer Science* **321**(2 – 3): 283 – 290.

Pandiselvam P., Marimuthu T. & Lawrance R. (2014). Comparative Study on String Matching Algorithms of Biological Sequences. arXiv:1401.7416 [cs.DS]

Shoshana N. (2009). Pattern Matching Algorithms: An Overview. Available at *www.sci.brooklyn.cuny.edu /~shoshana /pub/ secondExam.ppt*, Accessed 20 February 2017.

Van L.J. (1990). *Handbook of Theoretical Computer Science, volume A - Algorithms and Complexity*. MIT Press, Cambridge, MA, USA.

Vidanagamachchi S.M. (2017). Quantitative and qualitative computational pipelines for high-throughput shotgun proteomics. *PhD thesis*, University of Peradeniya, Peradeniya, Sri Lanka.

Vidanagamachchi S.M. (2017). Quantitative and qualitative computational pipelines for high-throughput shotgun proteomics. *PhD thesis*, University of Peradeniya, Peradeniya, Sri Lanka.

Vidanagamachchi S.M., Dewasurendra S.D., Ragel R.G. & Niranjan M. (2012). Commentz-Walter: any better than Aho-Corasick for peptide identification? *International Journal of Research in Computer Science* **2**(6): 33 – 37.

Watson B.W. (1994). The performance of single keyword and multiple keyword pattern matching algorithms. *Computing Science Notes*, volume 9419 (eds. J.C.M. Baeten & M. Rem). Eindhoven University of Technology, The Netherlands.

Watson B.W. (1995). Taxonomies and toolkits of regular language algorithms. *PhD thesis*, chapter 13, Eindhoven University of Technology, The Netherlands.