Review Article

iMedPub Journals
http://www.imedpub.com

American Journal of Computer Science and Information Technology

2021
Vol. 9 No. 9: 112

Evaluating Efficiency of Some Exact String-Matching Algorithms on Large-Scale Genome

Osamh Alrouwab[1*], Dheba Mansour[2] and Mahmoud Gargotti[3]

[1]Department of Biochemistry, Faculty of Medicine, University of Zawia, Zawia,Libya
[2]Department of Zoology, Faculty of Sciences, Aljafra University, Almamura, Libya
[3]Faculty of Medicine, Department of Microbiology, University of Zawia, Zawia, Libya

Corresponding author:
Osamh Alrouwab, Department of Biochemistry, Faculty of Medicine, University of Zawia, Zawia, Libya

usamaerawab@gmail.com

Citation: Alrouwab O (2021) Evaluating Efficiency of Some Exact String-Matching Algorithms on Large-Scale Genome. Am J Compt Sci Inform Technol Vol.9 No.9: 112.

Abstract

Exact string-matching algorithms have become very supreme in many bioinformatics tools. Despite the abundance and diversity of such algorithms, exposing them to real-time experimental analysis has been critical. This study was conducted to evaluate the efficiency of ten exact-string matching algorithms on large-scale genomic sequences from a runtime perspective. To define the most efficient algorithms are qualified to handle the short alphabet used for nucleic acid coding.

The methodology promoted for this study was the factorial experiment with Randomized Complete Block Design (FRCBD). Under influence of four independent parameters, four levels of pattern lengths, four levels of pattern indices, two levels of programming languages, and ten levels of algorithmic architecture. The yield of the tested algorithms was calculated in nanoseconds. One-way ANOVA and Two-way ANOVA tests with post-hoc Games-Howell test were used separately for statistical analysis. In this study two widely accepted programming languages, C# and JAVA were used to speculate the possible effect of programing language on algorithm performance.

The One-way ANOVA results revealed that the Backward-Oracle-Matching (BOM), Zhu-Takaoka (ZT), and Horspool's (HP) algorithms exhibited the highest final performance correspondingly. These algorithms have demonstrated an efficiency of up to 250% higher than other algorithms. The results of two-way ANOVA revealed a significant interaction between programing language adopted and execution time with the absence of pattern lengths and pattern index effect. The combination of the C# programing language and the Backward-Oracle-Matching algorithm produced the most effective performance on genomic sequences.

Keywords: Exact-string matching algorithm; Factorial design; One-way ANOVA; Two-way ANOVA; Games-Howell test

Received: September 24, 2021; **Accepted:** October 08, 2021; **Published:** October 15, 2021

Introduction

Admittedly, string-matching is an essential problem-solving technique, encountered by specialists from various disciplines e.g. Data mining, artificial intelligence, and Bioinformatics [1]. Oodles of algorithms and methods have been announced for pattern recognition, and there are abundant applications and online servers that can achieve precise string matching on biological data [2]. The methodologies that endorse the recognition of the patterns contrast greatly, owing to the obvious variations in algorithmic architecture [3]. Generally, string matching algorithms could be broadly classified into five distinct classes: (a) algorithms used to resolve the problem by character comparisons, (b) algorithms that depend on the use of automatic probabilistic simulation, (c) non-probabilistic simulation algorithms, (d) constant-space algorithms, and (e) real-time algorithms [4]. The more traditional and the humblest match approach are to equate the pattern characters with the characters in the target text. This implementation was offered by the so-called Naïve or Brute-Force algorithm [5]. Text or pattern has not been pre-processed by this algorithm. Its time complexity in the worst case is O (nm), where m and n apply to pattern and text length correspondingly. Subsequently, numerous algorithms have made formidable enhancements on Brute-Force time scheming. The worst-case, lower bound of the string-matching problem is O (n). The first algorithm to reach the bound was given by Morris and Pratt in the early [6] later improved by Knuth. Linear algorithms that are based on bit-parallelism were announced by Baeza-Yates and Manber Xian-Feng et al. presented the KMPBS algorithm, a hybrids algorithm based on Boyer–Moore (BM) and The Knuth-Morris-Pratt (KMP) algorithm. The text T is scanned from left to right for the given pattern P of length m. When searching, the very last character of P is compared to the corresponding character

of text T, and the KMP algorithm is then used to compare the remainder of the characters if there is a match [7-29]. Cao et al. [7] formulated a character-based string matching algorithm that computes the statistical likelihood of each English letter in the pattern string based on its unique position in the pattern string. To calculate the mathematical likelihood and dynamic condition of each character in the pattern string, the suggested methodology utilizes optimization based on a high decision. Hakak et al. announced a novel exact-string matching methodology published in a research paper entitled "A new split based searching for exact pattern matching for natural texts"[8-10]. In this technique, the assigned pattern is split into two chunks. To enhance the search strategy, only the second chunk of the pattern is searched using a brute-force methodology against the given text. When the second chunk of the pattern is detected, the first chunk of the pattern is directly mapped based on the location of the second chunk. The number of biology texts-based info collected these days is generally increased at a pace rising [11]. Hence, the answer to the question, what is the algorithm that can be relied upon from the perspective of reliability and productivity among this tremendous momentum of available algorithms become an imperative necessity. The methodology espoused for this experiment was the so-called Factorial experiment with FRCBD, to cope with the interaction between the predetermined factors [12]. This experiment was contemplated while building the Bioinformatics library. The question then what is the most suitable exact string-matching algorithm for invoking biological data. Due to time constraints at that period, the Boyer–Moore algorithm was implemented and that it was not possible to make a comparison between the algorithms. Strategically, the roadmap for this study was to assess the productivity and effectiveness of some exact-string matching algorithms from a time-consuming outlook. For reliability, four independent factors were adapted, namely the length and the index of the pattern, the algorithm type and the programing language. To estimate the anticipated effect of those factors on the performance of each algorithm separately under the same experimental conditions [13]. Finally, the resulted data interpreted statistically in an accurate and unbiased manner. More specifically, in the current study six hypotheses were examined: (a) *Hypothesis* 1: All the algorithms have equal run time on average, (b) *Hypothesis* 2: pattern length does not affect searching speed on average. (c) *Hypothesis* 3: Algorithm type and pattern length are independent or the real impact of interaction is not prevalent, (d) *Hypothesis* 4: pattern position does not affect searching speed on average, (e) *Hypothesis* 5: Algorithm type and pattern position are independent or the real impact of interaction is not prevalent, (f) *Hypothesis* 6: programing language does not affect searching speed on average. The main goal of this paper is to evaluate the performance of some exact-string matching algorithms in terms of processing time, as well as to measure the impact of some factors that may affect the search result on a limited alphabet used to encode Deoxyribonucleic Acid [14].

Materials and Methods

System participants

The dataset: The data on Clostridium botulinum strain DFPST0029 chromosome (accession ID: NZ_CP028842), has been retrieved from online, publicly accessible databases, the Entrez Gene databases from The National Center for Biotechnology Information, during January 29, 2021. The FASTA sequence file was equipped with 3858511 DNA base-pair and used as target text [15].

Search patterns: Sixteen randomized pattern groups were configured to represent the contrast in position and length levels to simulate realistic algorithm operating conditions **(Table 1)**. Firstly, to investigate the impact of sequence length on the profitability of the algorithm, the length was subsequently extended four times. The mean of the pattern length was 3500 (SD=3341.66). Lastly, to gauge the influence of the pattern site on algorithm run time, the pattern length was set to four positions that represented the topographic regions of the target text. The mean of the pattern position was 921542.60 (SD=994136.98) [16].

Table 1: Pattern sets.

Position ID	Position Index	Length ID	Pattern length
P1	0	L1	500 b.p
P2	286170	L2	1500 b.p
P3	1200000	L3	4000 b.p
P4	2200000	L4	8000 b.p

Software and operating system

The benchmarks have all been performed on Intel (R) Core (TM) i5-3470 CPU; 3.20 GHz and 16 GB DDR3 RAM. The operating system used for the benchmarks was Microsoft Windows 10 64-bit. Two compilers were recruited to measure the anticipated effect of the programing language on algorithm execution time. The C# from Microsoft using NET Framework 4.5.2 and JAVA(™) SE Development Kit 11.0.9 (JDK 11.0.9) from Oracle Corporation [17].

Measurements

Bio-statistical experimental design: The study was established in a Factorial randomized complete block design fashion. All algorithms were subjected to uniform conditions in terms of the input sequence, the patterns, and the hardware [18]. The design of the experiment encompassed four independent factors: the programing language (two levels; C# or JAVA); the exact string-matching algorithm used to detect search pattern (ten levels; Brute Force, Backward-Oracle-Matching, Raita, Horspool's, Rabin Karp, Berry-Ravindran, Zhu-Takaoka, Simon, Maximal-Shift or Two-Way Algorithm); the pattern position (four index levels; P1=0, P2=286170, P3=1200000 or P4=2200000); and pattern length (four levels; L1=500 b.p, L2=1500 b.p, L3=4000 b.p or L4=8000 b.p) resulting in a total of three hundred and twenty possible treatment combinations (N=320). Execution time for each algorithm measured in nanoseconds was assigned as the dependent response variable **(Figure 1)**.

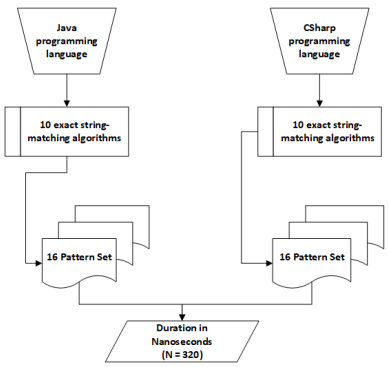

Figure 1 Flowchart of experiment workflow.

Statistical analyses of the data: All statistics were performed using the IBM SPSS 22 statistical package. To eliminate system interruptions all unnecessary running processes were halted, the same empirical restrictions were applied to all algorithms [19]. The data normality was tested using the Kolmogorov-Smirnov and Shapiro-Wilk tests. Means were compared using a one-way Analysis of Variance (ANOVA) and two-way ANOVA. Finally, a post-hoc test was conducted by the Games-Howell test **(Figure 2)**.

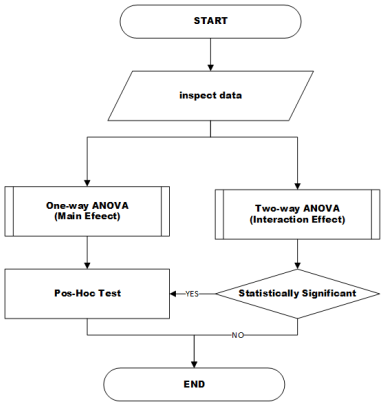

Figure 2 Flowchart of experiment statistical analyses.

Results

Interpretation of SPSS results

ANOVA table was determined by SPSS visualizing the p-value for the main effects (algorithm type, pattern position, pattern length, and programing language) and their interactions. An appropriate 95% Confidence Interval (CI) was given. A p-value of less than .05 implies a statistically significant main effect or effect of the interaction [20].

Preliminaries

This study intended to conduct a robust experiment, to assess the effectiveness of several exact-string matching algorithms under distinct variables. Ten exact-string matching algorithms were subject to unbiased tests. The data reported in this study have been achieved by a factorial RCBD experiment and subjected to statistical factorial analysis to measure the main effects of the four independent factors [21].

Effect of algorithm type on execution time

The descriptive statistics consorted with confidence intervals (CI 95%) across the ten algorithm type groups are proclaimed **(Table 2)**. As depicted the Backward-Oracle-Matching algorithm was linked with the numerically least mean level of execution time confidence (M=24253831) and the Berry-Ravindran algorithm was associated with the numerically most mean level of execution time confidence (M=65723113). To test *Hypothesis* 1: All the algorithms have equal run time on average; a between-groups ANOVA was performed. Preliminarily, the normality of data distribution must be fulfilled to conduct the ANOVA, a Kolmogorov-Smirnov test **(Table 3)** indicates that the means on trial follow a normal distribution, D (320)=0.145, p=.200. The assumption of homogeneity of variances was measured and fulfilled on Levene's F test F (9,310)=2.73, p=.004. The independency between-groups ANOVA produced a statistically significant effect, F (9,310)=2.60, p=.007, η_p^2= .007. Thus, all relevant conclusions are in favor of rejecting the H0 for *Hypothesis* 1. The performance affected by algorithm type and 7% of the variance in execution time was considered for by algorithm type membership **(Figure 3)**. To assess the nature of the variances between the means supplementary, the statistically significant ANOVA was followed-up with by the Games-Howell post-hoc tests. In order to check for individual difference between algorithm types post-hoc comparison using the Games-Howell test was selected. The results reveal that the mean score for the Brute Force algorithm (M=40777000, SD=9102098) was significantly diverged from Horspool's (M=32954493, SD=7978283), Rabin-Karp (M=65079287, SD=12222770), Backward-Oracle-Matching (M=24253831, SD=16582280) and Simon (M=51696978, SD=14748130) algorithm. The Horspool's (M=32954493, SD=7978283) was significantly diverged from Rabin-Karp (M=65079287, SD=12222770) and Simon (M=51696978, SD=14748130) algorithm [22]. The Zhu-Takaoka (M=31234328, SD=13230735) was significantly diverged from Rabin-Karp (M=65079287, SD=12222770) and Simon (M=51696978, SD=14748130) algorithm. The Raita was significantly diverged from Rabin-Karp (M=65079287, SD=12222770) and Simon (M=51696978, SD=14748130) algorithm. The Rabin-Karp (M=65079287, SD=12222770) was significantly diverged from Maximal-Shift (M=21891320, SD=42857463), Backward-Oracle-Matching (M=24253831, SD=16582280) and Simon (M=51696978, SD=14748130) algorithm. The Backward-Oracle-Matching (M=24253831, SD=16582280) was significantly diverged from Simon (M=51696978, SD=14748130) algorithm. The mean difference was significant at the .05 level. However, no significant difference reported between other group members [23].

Table 2: Performance results of tested algorithms.

	N	Mean	Std. Deviation	Std. Error	95% Confidence interval for mean		Minimum	Maximum	P4
					Lower bound	Upper bound			
BF(Backward Forward)	32	40777000	9102097.6	1609038.7	37495344	44058656	26000000	73000000	
HP(Horspool)	32	32954494	7978283.5	1410374.6	30078016	35830972	22000000	57000000	
BR(Break Key)	32	65723113	138603343	24501841	15751279	115694947	27000000	8.00E+08	
TW(Term Work)	32	55382887	69492399	12284637	30328206	80437569	29000000	4.00E+08	
ZT(ZhuTakaoka)	32	31234328	13230735	2338885.7	26464139	36004517	21000000	81000000	
RT(Run Time)		34778041	11037655	1951200.2	30798542	38757540	18000000	66000000	
RK(Radial Keratotomy)	32	65079287	12222771	2160701	60672509	69486066	24000000	77055899	
MS(Multiple Sclerosis)	32	42857463	21891320	3869875.1	34964800	50750125	13007900	1.00E+08	
BOM(Backward Oracle Matching)	32	24253831	16582280	2931360.7	18275282	30232381	12481500	82000000	
SMN	32	51696978	14748130	2607125.6	46379710	57014246	37000000	1.00E+08	
Total	320	44473742	51732458	2891932.3	38784072	50163412	12481500	8.00E+08	

Table 3: Tests of normality.

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Statistic	Df	Sig.	Statistic	Df	Sig.
Duration	0.145	320	.200*	0.946	320	0.427

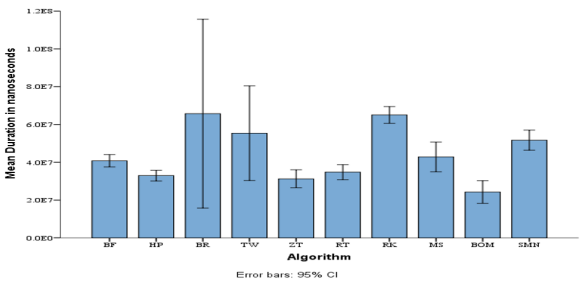

Figure 3 Performance for different algorithm types.

Correlation of algorithm types and pattern length

The experiment error rates have been forwarded to a two-way ANOVA with four levels of pattern lengths (L1, L2, L3, and L4) and ten levels of algorithm type. The outcomes designate that the ramifications of the algorithmic type performed a significant effect in the number of errors among suggested patterns, the results elucidated that 7.8% of variances in algorithms execution time was explicated by algorithm types (F (9,280)=2.62, p<.006, η_p^2=.078). The encouragement of pattern length, responsible only for 1% of variances in algorithms runtime as the result revealed (F (3,280)=0.85, p=.47, partial η_p^2 =.01). The cross-action between algorithm type and pattern length scores 10% of variance, (F (27,280)=1.10, p=.34, η_p^2 =.10). All relevant outcomes are in favor of accepting the H0, for *Hypothesis* 2 and *Hypothesis* 3. The pattern length does not affect the speed of the algorithm **(Figure 4)** and they are independent **(Table 4)** [24].

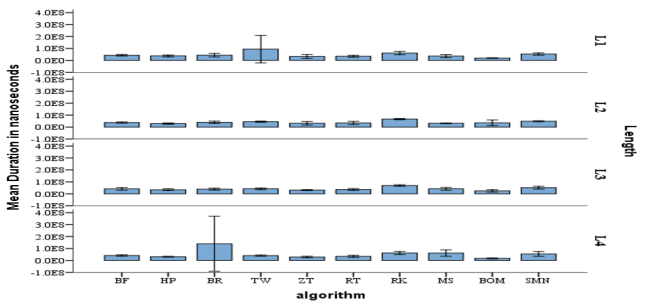

Figure 4 Impact of pattern length on algorithm run time.

Correlation of algorithm types and pattern position

A two-way analysis of variance discloses that the pattern position was statistically insignificant at P>.05. The effect of algorithmic type demonstrating that 8% of the variance in the algorithm

execution time was clarified by algorithmic type (F (9,280)=2.56, p=.008, η_p^2 =.08). The main impact of pattern position yielded an effect size of .01, divulge that the pattern location behind 1% of the variance in the algorithm execution time (F (3,280)=.73, p=.53, η_p^2 =.01). The interaction effect between the two factors was highly insignificant (F (27,280)=.84, p=.70, η_p^2 =.08), indicate that no significant combined effect was observed for algorithmic type and pattern position on algorithm execution time, responsible only for 8% of the variance [25]. All relevant outcomes are in favor of accepting the H_0, for *Hypothesis* 4 and *Hypothesis* 5. The pattern position does not affect the speed of the algorithm **(Figure 5)** and they are independent **(Table 5).**

Correlation of algorithm types and programing language

A two-way analysis of variance was conducted on the supremacy of two independent variables (algorithm type and programing language) on the execution time [26]. The algorithm type included ten levels (Brute Force, Backward-Oracle-Matching, Raita, Horspool's, Rabin Karp, Berry-Ravindran, Zhu-Takaoka, Simon and Maximal-Shift, Two-Way) and programing language consisted of two levels (C# and JAVA). All factors were statistically significant at the .05 significance level, except for the interaction between algorithm type and programing language. The main effect for the programing language yielded an F ratio of F (1, 300)=5.55, p=.02, indicating a significant difference between C# programing language (M=37895571, SD=16164128) and JAVA programing language (M=51051913, SD=70858720). The main effect for algorithm type yielded an F ratio of F (9, 300)=2.67, p=.005, indicating that the effect for algorithm type was statically significant, Brute Force (M=40777000.03, SD=9102097.57), Backward-Oracle-Matching (M=24253831.31, SD=16582280.02), Raita (M=34778040.63, SD=11037654.94), Horspool's (M=32954493.69, SD=7978283.54), Rabin Karp (M=65079287.44, SD=12222770.78), Berry-Ravindran (M=65723112.56, SD=138603342.72), Zhu-Takaoka (M=31234328.09, SD=13230735.42), Simon (M=51696978.19, SD=14748129.57), Maximal-Shift (M=42857462.50, SD=21891319.54) and Two-Way (M=55382887.44, SD=69492398.66). The interaction effect was insignificant, F (9, 300)=1.43, p=.041. All relevant outcomes are in favor of rejecting the H_0 and accepting H_1 for *Hypothesis* 6. The type of programing language has an impact on the execution time of the algorithm **(Figure 6)** without observed significant interaction between algorithm type and programing language **(Table 6)**.

Table 4: Independent variable: Algorithm type and pattern length.

Source	Type III sum of squares	df	Mean square	F	Sig.	Partial Eta squared
Algorithm	5.998E+16	9	6.665E+15	2.622	0.006	0.078
Length	6.51E+15	3	2.17E+15	0.854	0.466	0.009
algorithm × Length	7.555E+16	27	2.798E+15	1.101	0.338	0.096
Error	7.117E+17	280	2.542E+15			
Total	1.487E+18	320				
Corrected Total	8.537E+17	319				
Note: Intel Squared=.166 (Adjusted R Squared=.050)						

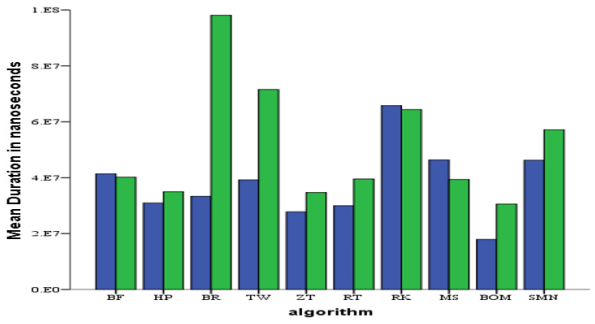

Figure 5 Impact of pattern position on algorithm run time.

Table 5: Independent variable: Algorithm type and pattern position.

Source	Type III sum of squares	df	Mean square	F	Sig.	Partial Eta squared
algorithm	5.998E+16	9	6.665E+15	2.56	0.008	0.076

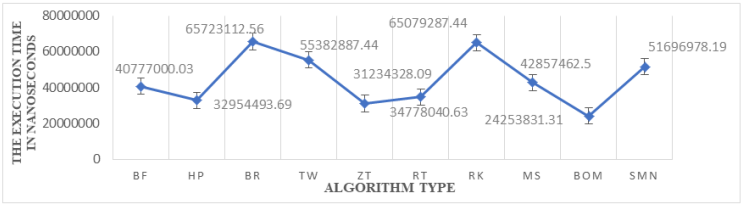

Figure 6 Impact of programing language and algorithm type.

Table 6: Independent variable: Algorithm type and programing language.

Source	Type III sum of squares	df	Mean square	F	Sig.	Partial Eta squared
algorithm	5.998E+16	9	6.665E+15	2.56	0.008	0.076
Programing language	1.385E+16	1	1.385E+16	5.554	0.019	0.018
Algorithm	5.998E+16	9	6.665E+15	2.673	0.005	0.074
Programing language × algorithm	3.2E+16	9	3.556E+15	1.426	0.176	0.041
Error	7.479E+17	300	2.493E+15			
Total	1.487E+18	320				
Corrected Total	8.537E+17	319				

Note: Intel Squared=.124 (Adjusted R Squared=.068)

Discussions

This study approached the problem from the exact-string matching factor of perspective. To make a definitive distinction between the productivity of frequently accepted exact string-matching algorithms on nucleotide alphabet. Essentially, throughout molecular investigations, scanning for oligonucleotides patterns was considered a commonly performed task. DNA antisense, microarray, gene cloning, and polymerase chain reaction analyses all need to be performed a string-matching in one form or another. Constructing an application based on reliability, productivity, and suitability for genomic sequences requires distinctiveness between available algorithms and selecting the best.

The outcome of algorithm design on runtime

The influence of algorithm architectural design on the runtime of exact-string matching algorithms was widely discussed in prior studies [27]. According to Christian and others reported that the Boyer-Moore-Horspool algorithm which does preprocess on search patterns performed better on short patterns than the naïve algorithm which lacks the preprocessing step [28]. Simone and Thierry addressed the exact string-matching problem in an elaborate experiment; their results reveal that for various alphabet sizes and pattern lengths the efficiency of algorithms is quite diverse [29]. Furthermore, AbdulRazzaq concluded the impact of algorithm architecture was the cornerstone that affected the performance of some exact string-matching algorithms. It is obvious from the related literature reviews, that the algorithm architecture appears to have an influential role in performance at the time of implementation. The awaited significance of algorithm design was formulated as a *Hypothesis* 1 in this study. The best performance in the current study was scored by the Backward-Oracle-Matching algorithm **(Figure 7)**. It's an automaton on a word p, the sequence of letters taken in an alphabet Σ, that combination called factor oracle. The Two-Way algorithm ranked second in terms of performance. It's a variant of the Boyer-Moors algorithm. The rest of the algorithms have a fairly close performance, except for both the Berry-Ravindran and the Rabin-Karp respectively had a poor performance [30]. The results recorded in this experiment are consistent with past findings, which prove the existence of an effect of algorithm design on runtime.

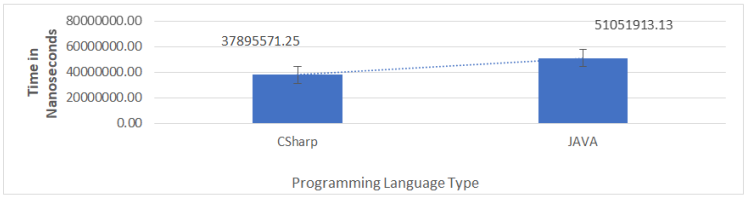

Figure 7 Run time for evaluated algorithms.

The outcome of the programming languages on runtime

To our understanding, this is the first study to evaluate the effect of the programming language type on run time for the exact string-matching algorithms. A *hypothesis* was verbalized, *Hypothesis* 6. To estimate the probable consequence of a programming language type on the algorithm's run time. The results gained were in courtesy of accepting the alternative *hypothesis*. The C# provides better performance than JAVA superior by 75%.

Challenges

Monitoring the efficiency of exact string-matching algorithms in terms of performed tasks (e.g. palindrome sequence, and fingerprint detection) and categorizing them by productivity rather than the methodologies used are challenging. However, putting the focus on particular tasks assists the researcher to improve or implement only specific algorithms instead of randomly selecting the algorithms.

Conclusions

In this study, the fastest algorithms were Backward-Oracle-Matching, Zhu-Takaoka, and Horspool's respectively. The architecture of the algorithm plays a critical role in the performance. Moreover, the C# programming language provided an outstanding performance superior to the Java language and verified that the programming language has an effective role in the run time of the algorithms under trial. No pattern-related influence has been shown, either on the length of the pattern or on its positioning on the target text, as contrasted to any previous studies that indicate the remarkable effect of this factor. Finally, we strongly recommended adding new algorithms to evaluate their performance. Additionally, expanding the scope of the possible factors that may interfere with the performance of algorithms run time, such as the operating system and the alphabet in future studies.

References

1 Abbott A, Tsay A (2000) Sequence analysis and optimal matching methods in sociology. Sociol Methods Res 29: 3-33.

2 Razzaq AA, Rashid NA, Hasan AA, Hashem MA (2013) The exact string matching algorithms efficiency review. Glob J Technol 9: 12-18.

3 Allauzen C, Crochemore M, Raffinot M (1999) Factor Oracle: A New Structure for Pattern Matching. International Conference on Current Trends in Theory and Practice of Computer Science 27: 295-310.

4 Zhang C, Pang J (2012) An Algorithm for Probabilistic Alternating Simulation. International Conference on Current Trends in Theory and Practice of Computer Science 21: 431-442.

5 Boyer RS, Moore JS (1977) A Fast String Searching Algorithm. Communications of the ACM 20: 762-772.

6 feng HX, Yubao YY, Lu X (2010) Hybrid Pattern-Matching Algorithm based on BM-KMP Algorithm. International Conference on Advanced Computer Theory and Engineering 8 :305-310.

7 Cao Z, Yan Z, Liu L (2015) A Fast String Matching Algorithm based on Lowlight Characters in the Pattern. International Conference on Advanced Computational Intelligence (ICACI) 27: 179-182

8 Hakak S, Kamsin A, Shivakumara P, Idris MYI, Gilkar GA (2018) A new split based searching for exact pattern matching for natural texts. 13: 24-26

9 Krallinger M, Valencia A, Hirschman L (2008) Linking genes to literature: text mining, information extraction, and retrieval applications for biology. Genome Biol 9: 1-4.

10 Allmer J (2017) Exact pattern matching: Adapting the Boyer-Moore algorithm for DNA searches. PeerJ PrePrints.

11 Berry T, Ravindran S (1999) A Fast String Matching Algorithm and Experimental Results. In Stringology 19: 16-28.

12 Washietl S (2005) Prediction of structural non-coding RNAs by comparative sequence analysis.

13 AL970861RF MC, Perrin D (1991) Two-way string-matching. J Assoc Comput 38: 651-675.

14 Deighton RA. Using Rabin-Karp fingerprints and Level DB for faster searches.

15 Frakes WB, Yates RB (1992) Information retrieval: Data structures and algorithms. Prentice-Hall.

16 Knuth DE, Morris, Jr JH, Pratt VR (1977) Fast pattern matching in strings. J Comput 6: 323-350.

17 Michailidis PD, Margaritis KG (2002) On-line approximate string searching algorithms: Survey and experimental results. Int J Comput Math 79: 8678-88.

18 Morris Jr J, Pratt V (1970) A linear pattern-matching algorithm.

19 Mozgovoy M (2007) Enhancing computer-aided plagiarism detection. Joensuun yliopisto.

20 Naser MA, Rashid NA, Aboalmaaly MF (2012) Quick-skip search hybrid algorithm for the exact string matching problem. Int J Comput Theory Eng 4:259-262.

21 Raita T (1992) Tuning the boyer-moore-horspool string searching algorithm. Software: Practice and Experience 10:879-884.

22 Rasool A, Tiwari A, Singla G, Khare N (2012) String matching methodologies: A comparative analysis. 11: 30-40.

23 Sahota V, Li M, Bayford R (2013) MPS: improving exact string matching through pattern character frequency. J Data Process 3: 127-129.

24 Sheik SS, Aggarwal SK, Poddar A, Sathiyabhama B, Balakrishnan N et al. (2005) Analysis of string-searching algorithms on biological sequence databases. J Curr Sci 25:368-374.

25 Simon I (1994) String matching algorithms and automata. Trends Theor Comput Sci 23: 386-395.

26 Lovis C, Baud RH (2000) Fast exact string pattern-matching algorithms adapted to the characteristics of the medical language. J Am Med Inform Assoc 7: 378-391.

27 Faro S, Lecroq T (2010) The exact string matching problem: a comprehensive experimental evaluation.

28 Sunday DM (1990) A very fast substring search algorithm. Commun ACM 33:132-142.

29 Wu S, Manber U (1992) Agrep–A Fast Approximate Pattern-Matching Tool. In Usenix Winter Technical Conference 24: 153-162.

30 Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn 42: 31-60.