A New Efficient Hybrid Exact String Matching Algorithm and Its Applications

Atheer Akram AbdulRazzaq, Nur'Aini Abdul Rashid, Muhannad A. Abu-Hashem, Awsan Abdulrahman Hasan

Department of Parallel and Distributed Processing, School of Computer Sciences Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia
athproof@yahoo.com

Abstract: String matching is one of most challenging issues in computer science. In this study, a new efficient hybrid string matching algorithm called Atheer was developed. This proposed algorithm is integrated with the excellent properties of three algorithms, namely, the Karp–Rabin, Raita, and Smith algorithms. The Atheer algorithm demonstrated an efficient performance in the number of comparison attempts as well as in the character comparisons with original algorithms in the first step and with recent and standard algorithms (i.e., Horspool, Quick search, Two-way, Fast search, SSABS, TVSBS, AKRAM, and Maximum shift) in the second step. The proposed algorithm in this study utilized several data types, namely, DNA sequences, Protein sequences, XML structures, Pitch characters, English texts, and Source codes. The Pitch database was the best match for Atheer in terms of the number of comparison attempts involving long and short patterns; the DNA database was the worst match. In terms of the character comparisons, the best database was the Source code database; the DNA sequence data type was also the worst match when short and long patterns were used.
[AbdulRazzaq AA, Rashid NA, Abu-Hashem MA, Hasan AA. **A New Efficient Hybrid Exact String Matching Algorithm and Its Applications.** *Life Sci J* 2014;11(10):474-488] (ISSN:1097-8135).
http://www.lifesciencesite.com. 65

Keywords: Exact string matching, Atheer algorithm, type and size of data, pattern lengths

1. Introduction

String matching is the process of finding all occurrences of alignments by comparing two finite-length strings (Faro and Lecroq, 2013). It is one of the most challenging issues in many computer science applications, including operating systems, information retrieval from databases, web search engines (Bhukya and Somayajulu, 2011), intrusion detection systems (Hassan and Rashid, 2012), signal and image processing (Lu, 2008; Klaib and Osborne, 2009), artificial intelligence (Al-mazroi and Rashid, 2011), compilers, and command interpreters. Other examples of string matching applications are library systems, error correction, text processing, speech and pattern recognition (Michailidis and Margaritis, 2002), bibliographic search, question-answer applications (Zubair et al., 2010), and in the literature of dictionaries and memorized data (Hassan, 2005). String matching is also used to analyzed Protein sequences and pattern matching of DNA (Cao, 2004; Bhukya and Somayajulu, 2011). Therefore, string matching plays a significant role by inducing challenging problems in theoretical computer science (Hassan, 2005; Faro and Lecroq, 2010).

String matching involves patterns and texts, both of which undergo a matching process to identify their identical characters. The matching process depends on two factors: the number of characters compared and the number of attempts made for the comparison. These factors are changeable depending on the type of algorithm used (Lecroq, 1995; Kadhim, 2012; Hussain et al., 2013). In this paper, the proposed new hybrid algorithm depends on the good features of existing exact strings matching algorithms and overcome the disadvantages of these algorithms which are Karp-Rabin, Smith and Raita. The proposed hybrid algorithm uses all types of databases existed in benchmark standard to determine the suitable and unsuitable databases for this algorithm. The main aim of this paper is to improve the performance of existing string matching algorithms.

1.1Original algorithms

The demand for efficient hybrid exact string matching algorithms has increased because of the need to minimize the limitations of the original algorithms and to obtain the best performance results (Abdulrazzaq et al., 2013a). Three original algorithms are used in the present study: Karp–Rabin, Smith, and Raita.

The Karp–Rabin algorithm is based on the hashing approach (Karp and Rabin, 1987) and depends on the matching process of the hashing function. In this function, each character in the string changes to the integer number that can facilitate transactions within a mathematical operation. The comparison in the searching phase begins with the comparison of the hash number of the pattern with the hash number of the text window. The shift in the pattern begins in the left and ends in the right. When a mismatch occurs, the window shifts to the right by one character. Rehashing is required to calculate the

new hash of the window after removing the hash value of the first character from the left side and after adding the new character from the right side in the previous step. The process of rehash continues up to the last character for every shifting process. When the hash number of the pattern equals the hash number of the text window, the characters in the pattern are compared with those of the text window one by one. After matching all characters, the pattern is moved by shifting one character. The hash function technique is considered a high-performance function because it uses integer numbers that reduce the computing time (Abdulrazzaq et al., 2009).

The Raita algorithm belongs to the Boyer–Moore subgroup (Raita, 1992), and its techniques are characterized by matching from any order behavior (Charras and Lecroq, 2004). This algorithm depends on the Boyer–Moore bad character (bmBc) table in the preprocessing phase. In the searching phase, the matching process starts after the rightmost character in the pattern is compared with the rightmost character in the text window. If a match is found, then the leftmost character in the pattern is compared with the leftmost character in text window. If the character also matches, then they are compared with the middle character in both the pattern and the text window. If a mismatch is found for each of the three characters, then the shifting of the pattern will depend on the *m* value in the bmBc table. If a match is found, the comparison starts from the second to the last character *(m−1).* The middle character is compared again during this process. If a mismatch occurs, then the shifting of the pattern will depend on the *m* value in the bmBc table. If a match is found, then the comparison continues to another character; when all the characters are reached, the shifting will depend on the *m* value (Abdulrazzaq et al., 2013b).

The Smith algorithm is a type of hybrid algorithm (Smith, 1991) whose technique is characterized by matching from any order behavior. This algorithm consists of two algorithms, namely, Horspool and Quick search. The preprocessing phase depends on the bmBc table and the quick search bad character (qsBc) table. The matching operation starts after the pattern and the text are compared from left to right. If a mismatch occurs, then the shifting of the pattern will depend on the higher value obtained in the comparison between the *m* value in the bmBc table and the *m + 1* value in the qsBc table. If a match is found, then the comparison continues to another character. If all the characters match, the shifting will then depend on the higher value between the *m* value in the bmBc table and the *m + 1* value in the qsBc table (Charras and Lecroq, 2004).

2. Method

The proposed algorithm, Atheer, depends on the preprocessing and searching phases and is integrated with the excellent features of the Karp–Rabin, Raita, and Smith algorithms.

2.1 Preprocessing phase

The preprocessing of this algorithm involves three steps:

(1) bmBc step

The technique in this step is similar to that in the two other original algorithms. This step is utilized in the preprocessing phase of the Raita and Smith algorithms and is employed in the hybrid algorithm as the initial step to prepare the bmBc table.

(2)qsBc step

This step is employed in the preprocessing phase of the Smith algorithm, with the Atheer algorithm using the same technique. The Atheer algorithm needs to prepare the qsBc table, which is the second step in the preprocessing phase of the hybrid algorithm. The best shifting process for each character from the bmBc and qsBc tables is selected for the hybrid algorithm.

(3)Hashing step

This step is derived from the preprocessing phase of the Karp–Rabin algorithm that depends on the hashing process (Figure 1). The technique in the Atheer algorithm differs from that in the original algorithm. This difference is related to the hashing characters in the Atheer algorithm, which calculates the first hashing step (Fh) in the pattern and the text window (Fhw). The second hashing step (Sh) and the third hashing step (Th) depend on the specific hashing value (Figure 1). Each letter used in the Atheer algorithm follows ASCII representation.

In the Fh step, hashing is calculated for only three characters, which are the last, first, and middle characters. The calculation depends on the equation number (1) for this step. The hashing of these three characters in the text window is calculated by the equation number (1) denoted by Fhw. In the Sh step, hashing is calculated from the second character to the *middle −1* character in the pattern. The calculation also depends on the equation number (2) for this step. In the Th step, hashing is calculated from the *middle +1* character to the *last −1* character in the pattern. The calculation also depends on the equation number (3) for this step (Th). For all the hashing steps, suppose that W_F, W_S, and W_T are found in Fh/Fhw, Sh, and Th, respectively.

```
1.  //(Fh)calculate the hash values of first step in pattern
2.   fhx ← (fhx<<1) + firstCh,  fhx ← (fhx<<1) + middleCh,  fhx ← (fhx<<1) + lastCh
3.  //(Fhw) calculate  the hash values of first step in text window
4.   fhy ← (fhy<<1) + y[0],   fhy ← (fhy<<1) + y[m/2],  fhy ← (fhy<<1) + y[m-1]
5.  //(Sh) calculate the hash values of second step
6.  shfx ← gethy(1, m/2, x)
7.  // (Th) calculate the hash values of third step
8.   shlx ← gethy (m/2+1, m-1, x)
```

Figure 1. Hashing process for the Atheer algorithm

The hashing value is calculated using the following equations:

First hashing step: $(w_F[0,m/2,m-1]) = (w_F[0] \times 2^{u-1} + w_F[m/2] \times 2^{u-2} + w_F[m-1] \times 2^0) \bmod q.$ (1)

Second hashing step: $(w_S[1 \ldots m/2-1]) = (w_S[1] \times 2^{u-1} + w_S[2] \times 2^{u-2} + \ldots + w_S[m/2-1] \times 2^0) \bmod q.$ (2)

Third hashing step: $(w_T[m/2+1 \ldots m-2]) = (w_T[m/2+1] \times 2^{u-1} + w_T[m/2+2] \times 2^{u-2} + \ldots + w_T[m-2] \times 2^0) \bmod q.$ (3)

The pseudo code for all the steps in the preprocessing phase of the hybrid algorithm is shown in Figure 2.

```
1.   Algorithm Atheer (X [0 .....m -1]
2.        //Input: Pattern X
3.        //Output: Shift tables of (bmBc), (qsBc) and compute the hush values.
4.        // preqsBc (preprocessing Quick-Search bad-character function)
5.        For K ← 0 to size of alphabet Do
6.             qsBc[k] ← m +1
7.        End For
8.        For j ← 0 to m-1 Do
9.             qsBc [X[j]] ← m- j
10.       End For
11.       // prebmBc (preprocessing Boyer-Moore bad-character function)
12.       For K ← 0 to size of alphabet Do
13.            bmBc [k] ← m
14.       End For
15.       For j ← 0 to m -2 Do
16.            bmBc [X[j]]←  m- j -1
17.       End For
18.       // Compute the hush values h = d^S-1 mod q
19.       For  i ← w  to S-1 Do
20.            hy ← (hy<<1)+y[i]
21.       End For
22.       firstCh ← x[0], secondCh ← x+1, middleCh ← x[m/2], lastCh ← [m-1]
23.       // Hash values of all steps in pattern and the first three characters in text window
24.       fhx ← (fhx<<1) + firstCh,  fhx ← (fhx<<1) + middleCh,  fhx ← (fhx<<1) + lastCh
25.       fhy ← (fhy<<1) + y[0],   fhy ← (fhy<<1) + y[m/2],  fhy ← (fhy<<1) + y[m-1]
26.       shfx ← gethy(1, m/2, x)
27.       shlx ← gethy (m/2+1, m-1, x)
```

Figure 2. Preprocessing phase in the Atheer algorithm

2.2 Searching phase

The searching phase technique in the proposed algorithm depends on the searching phase techniques of the original algorithms and on some of the modulations during the matching operation. The first step compares the hash values of the three characters in the Fh pattern with the hash values of the three characters in Fhw. If a match is found, then the three characters in the text window and the three characters in the pattern are compared one by one. If a match is found between these characters, then the second step is performed. If a mismatch occurs in the hashing comparison or in the character comparisons, then the shifting will depend on the maximum value

between *m* and *m + 1*, where *m* refers to the last character in the text window, and *m + 1* refers to the first character after the text window. This case is derived from the Smith algorithm and depends on the *m* value in the bmBc table and the *m + 1* value in the qsBc table.

If a match is found in the first step, then the hashing characters (Sh) in the pattern are compared with the hashing characters from the second to the *middle −1* characters in the text window in the second step. The hashing of the second step characters in the text window are calculated by using the equation number (2) denoted by (Shw). If a match is found, then the characters between them are compared. Regardless of whether a match or a mismatch is found, the shifting process will depend on the same technique used in the previous step. If a match is found in the second step, then the hashing characters (Th) in the pattern are compared with the hashing characters from the *middle character +1* to the *last −1* character in the text window in the third step. The hashing of the third step characters in the text window are calculated by using the equation number (3) denoted by (Thw). If a match is found, then the characters are compared. Regardless of whether a match or a mismatch is found, the shifting process will depend on the same technique used in the previous steps. The pseudo code for the steps in the searching phase of the Atheer algorithm is shown in Figure 3.

```
1.  Algorithm Atheer (X [0 .....m -1], Y [0…….n-1])
2.    //Input: Pattern X, Text Y
3.    //Output: number of attempts and number of character comparisons of pattern with text
4.    If (m%2 == 0) Then
5.        par ← 1
6.    End If
7.        j ← 0
8.    While j <= n – m Do
9.        c ← y[j + m - 1]
10.       // Comparing the Fh and Fhw
11.       If (fhx == fhy && lastCh == c && firstCh == y[j]&& middleCh== y[j + m/2])Then
12.           shfy ← gethy(j + 1, j + m/2, y)          //calculate the hash of (Shw)
13.           // Comparing the Sh and Shw
14.           If (shfx == shfy && match(x + 1, m/2 - 1, y, j + 1, &temp) == 1) Then
15.               shly ← gethy(j+m/2+1, j + m -1, y)   // calculate the hash of (Thw)
16.               // Comparing the Th and Thw
17.               If(shlx == shly && match(x + m/2 + 1, m/2-1-par, y, j + m/2 + 1, &temp) == 1)Then
18.                   Count          // The first occurrence of the pattern in the text
19.               End If
20.           End If
21.       End If
22.       Output the first attempt and character comparisons
23.       j +=max(qsBc[y [j + m]],bmBc[y[j + m - 1]])
24.       // Rehash operation for the text window
25.       fhy ← 0,fhy ← (fhy<<1) + y[j],fhy ← (fhy<<1) + y[j+m/2],fhy ← (fhy<<1) + y[j+m-1]
26.   End While
```

Figure 3. Searching phase in the Atheer algorithm

2.3 Proposed algorithm analyses

The time complexity of the bmBc and qsBc functions is denoted as *O (m + σ)*, and the hash function is denoted as *O (m)*. The time complexity of the preprocessing phase of the proposed algorithm is denoted as *O (m + σ),* and the space complexity is denoted as *O (σ).* The next section describes the time complexity of the searching phase.

Lemma 2.1. The time complexity of the search space in the best case is *O (n / (m + 1)).*

Proof. When each character during the comparison does not occur in the pattern, then the shifting depends on the maximum value between *m* and *m + 1*, where *m* is from the bmBc function, and *m + 1* is from the qsBc function; both values are computed during the preprocessing phase. When all characters in the pattern differ from the characters in the text, the shifting depends on *m + 1*, and the time complexity is *O (n / (m + 1)).*

For example:
Text: aaaaaaaaaaaaaaaaa
Pattern: bbbb
Lemma 2.2. The time complexity of the searching phase is *O (n × m)* in the worst case.

Proof. For each character in the text, the matching process does not take place more than *m* times. Thus, all the comparisons of *n* characters do not exceed *m × n*. The worst case occurs when all characters in the pattern and the text window are quite similar in every attempt. In this case, the shifting is equal to *1*, and the time complexity is *O (n × m)*.
For example:
Text: ccccccccccccccccc
Pattern: cccc

Given the size of the alphabet characters and the possibility of the appearance of every character in the text, the average time complexity is not determined in this algorithm.

2.4 Comparison of proposed and original algorithms

The comparison step shows the difference between the Atheer algorithm and the original algorithms. This difference is determined in terms of the preprocessing phase, searching phase and shifting operation (Table 1).

Table 1. Comparison of the Atheer algorithm and the original algorithms

Properties	Karp-Rabin	Raita	Smith	Atheer
Preprocessing phase				
Hashing	✓	×	×	✓
Using of Boyer-Moore bad character (bmBc) table	×	✓	✓	✓
Using of Quick search bad character (qsBc) table	×	×	✓	✓
Searching phase				
Hash comparisons of three characters(Fh and Fhw)	×	×	×	✓
Hash comparison of (Sh and Shw)	×	×	×	✓
Hash comparison of (Th and Thw)	×	×	×	✓
Pattern shifting by Maximum value between M and M+1	×	×	✓	✓
Other properties				
Comparison movement	Left to right	By any order	By any order	By any order
Shifting after matching	One character	Character value in bmBc table	Maximum value between m and m+1	Maximum value between m and m+1
Shifting after mismatching	One character	Character value in bmBc table	Maximum value between m and m+1	Maximum value between m and m+1

3. Experimental design

The design of hybrid algorithm depended on choosing the good properties of original algorithms and reformulating the searching phase of Atheer algorithm, which can be dealt with different benchmark standard databases. The hybrid algorithm was compared to original algorithms, and then to the recent and standard algorithms. The last step in this research was analyzing and evaluating the results of algorithm.

3.1 Databases

Common types of databases use string matching algorithms. The data types utilized in this study are DNA sequences, Protein sequences, XML structures, Pitch characters, English texts, and Source codes. These data types are considered the benchmark standard, and all were downloaded from the Pizza & Chili Corpus website (http://pizzachili.dcc.uchile.cl/ (Pizza Chili Corpus).

The DNA sequence data contained four nucleotides of DNA: adenine (A), guanine (G), cytosine (C), and thymine (T). The data were downloaded from Project Gutenberg (Karkkainen and Joong, 2006). The Protein sequence data composed of amino acid sequences were obtained from the Swissprot database. The XML structure text database included bibliographic information in the field of computer science. The Pitch character (MIDI Pitch values) data type specifies tuning data in digital music (Chew and Chen, 2003). The English text data included all the alphabet characters in the English language and were obtained from Project Gutenberg (Karkkainen and Joong, 2006). The Source code data were composed of all characters used in the C/Java languages (Ferragina and Fischer, 2007). The patterns lengths utilized in this study were divided into two types: short and long patterns. The short pattern length is between 4 and 28 characters, whereas the long pattern length (length power of two) ranges from $2^{\wedge 5}$ to $2^{\wedge 10}$. These types have been used in previous studies, and the characters of the patterns are randomly selected from the text (Huang et al., 2008; Cai et al., 2009). The total data size used in the present study was 50 MB.

3.2 Implementation and environment

The experiment was run on the Al Biruni cluster in the PDCC lab of the School of Computer Science, USM (Biruni.cs.usm.my), using Ubuntu Linux 10.04, 4LTS of 64-bit with NVIDIA CUDA Toolkit v2.2 and GNU C Compiler (GCC) v4.4.3. The secure shell (SSH) software was utilized in accessing the Biruni cluster to implement the codes.

The following abbreviated forms of the algorithms were used: R for Raita, K–R for Karp–Rabin and S for Smith. These algorithms are considered the original algorithms. The recent and standard algorithms included Horspool (H), Quick search (QS), Two-way (T-W), Fast search (FS), SSABS (SS), TVSBS (TV), AKRAM (AK), and Maximum shift (MS). Atheer (AT). To refine the results in the number of attempts made when the proposed hybrid algorithm was compared with the original, recent, and standard algorithms, the following parameters were set: logarithmic scale and base, 10; display units, 10000; minimum number, 100000. In the number of characters compared for the original, standard, and recent algorithms, the logarithmic scale and base (10) and the display units (10000) were applied. The results of this study were deemed superior when the results of the hybrid algorithm were better than those of the original algorithms. Thus, the tables of evaluation for the hybrid algorithm were arranged with the best result presented first, followed by the results from the other algorithms. The results were expressed as "first," "second," "third," and "fourth." To evaluate the performance of the hybrid algorithm in the different types of databases, the result was expressed as "best" when the hybrid algorithm performed better in a specific database compared with others. "Worst" was used to indicate the lowest performance of the hybrid algorithm for a given database. "All databases" was used to indicate when the hybrid algorithm or the other algorithms attained the best performance in all databases. "Most databases" was used to indicate that the hybrid algorithm or the other algorithms attained the best performance in most but not all databases.

4. Results

The comparison of the results of the Atheer algorithm with those of the original algorithms in the first step as well as with those of the recent and standard algorithms in the second step depends on the number of comparison attempts and characters compared. The performance depends on the data type and pattern lengths. All types of data utilized in this study (i.e., DNA sequences, Protein sequences, XML structures, Pitch characters, English texts, and Source codes) used a data size of 50 MB. The short (4 to 28) and long ($2^{\wedge 5}$ to $2^{\wedge 10}$) pattern length types were also used.

4.1 Results of the comparison of the Atheer algorithm with the original algorithms

The number of comparison attempts and characters compared required a data size of 50 MB. In the number of attempts, the Atheer and Smith algorithms achieved the best results compared with the Raita and Karp–Rabin algorithms in both short and long pattern lengths. The Pitch database showed the best results in terms of the number of comparison attempts when the long and short patterns were used; the DNA database demonstrated the worst results (Figures 4 and 5).

In the number of characters compared, the Karp–Rabin algorithm showed the best results in all short pattern lengths, except when a very short pattern length, such as 4. This algorithm was followed by the Atheer, Smith, and Raita algorithms. In terms of the long pattern length, the Karp–Rabin algorithm demonstrated the best results, followed by the Atheer, Smith, and Raita algorithms. The Source database showed the best results in terms of the number of characters compared when using short and long patterns, whereas the DNA database showed the worst results (Figures 6 and 7).

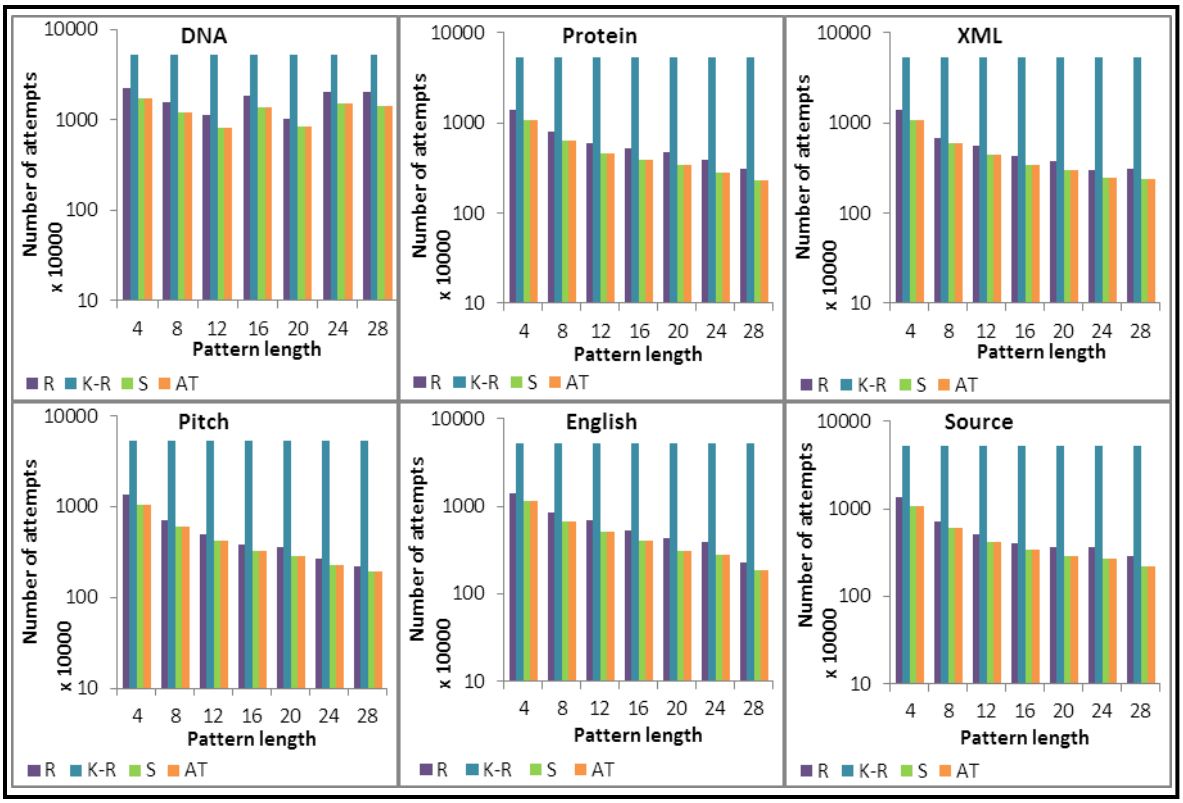

Figure 4. Number of attempts for the Atheer and the original algorithms when using a short pattern length and a database size of 50MB

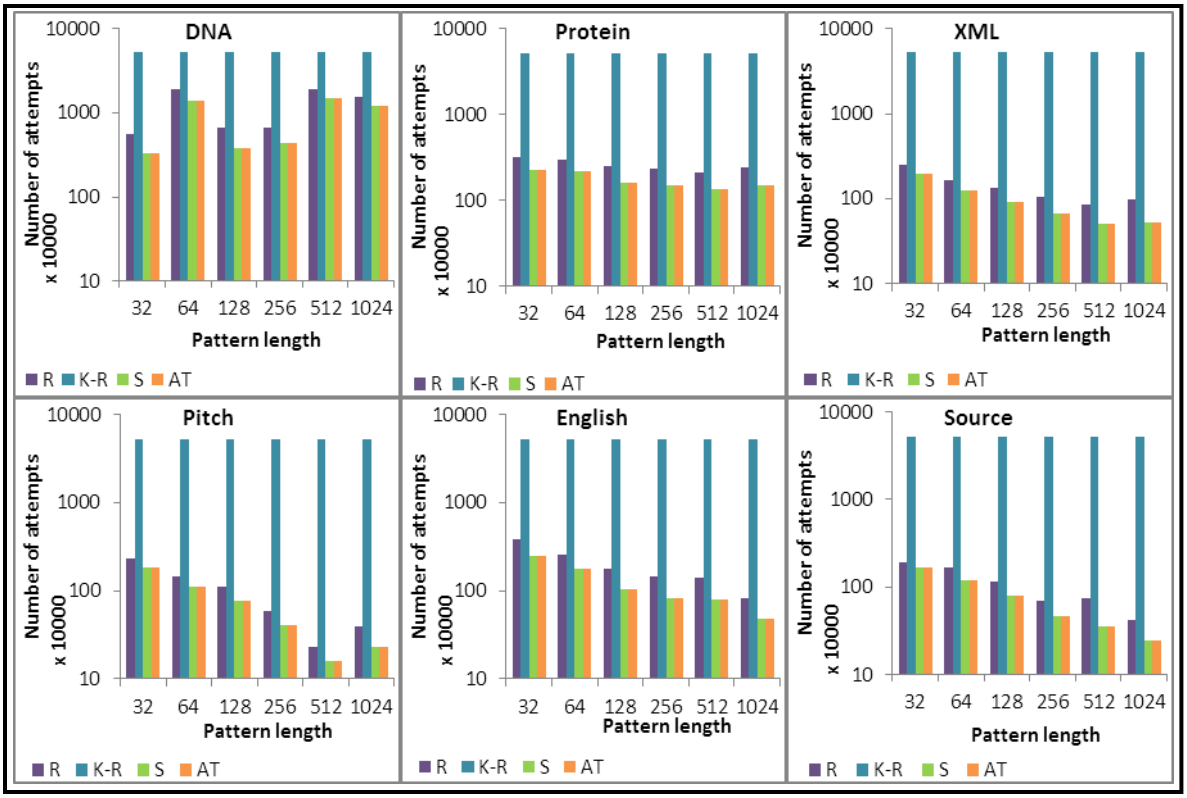

Figure 5. Number of attempts for the Atheer and the original algorithms when using a long pattern length and a database size of 50MB

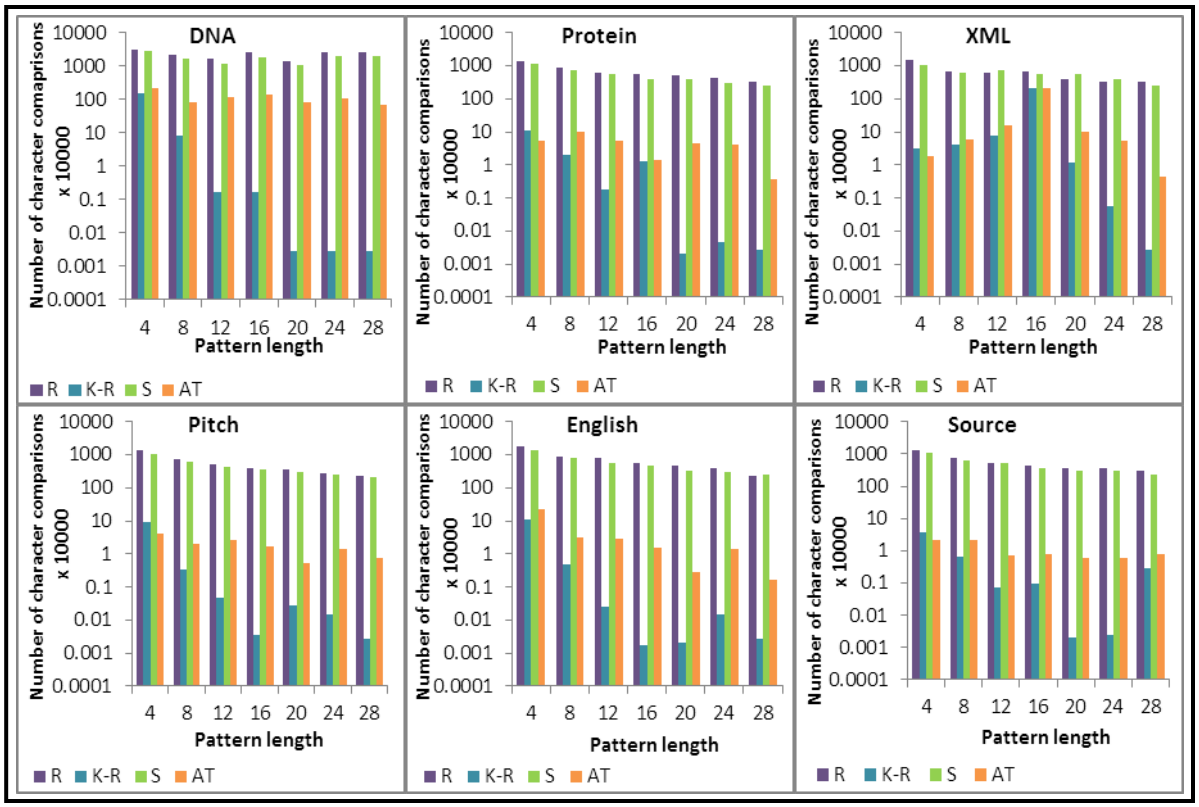

Figure 6. Number of characters compared for the Atheer and the original algorithms when using a short pattern length and a database size of 50MB

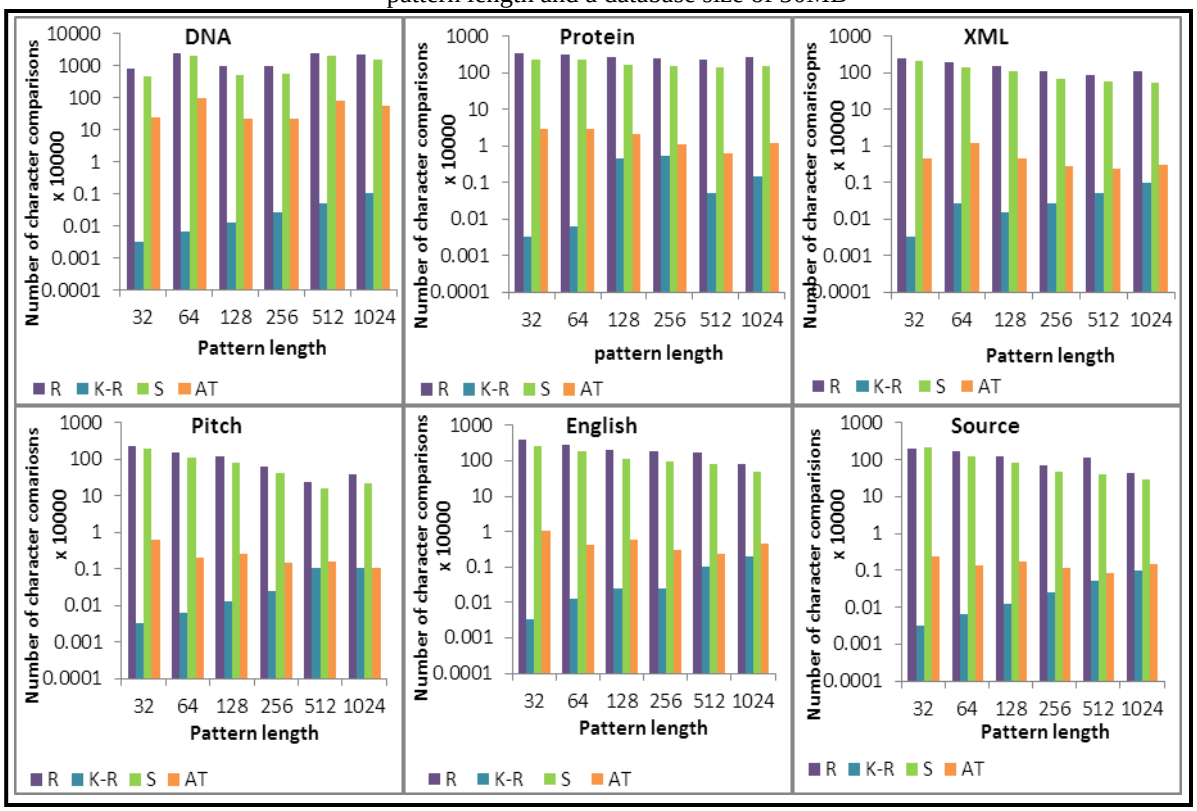

Figure 7. Number of characters compared for the Atheer and the original algorithms when using a long pattern length and a database size of 50 MB

4.2 Results of the comparison of the Atheer algorithm with recent and standard algorithms

In terms of the number of comparison attempts, the TVSBS showed the best performance in most databases when a short pattern length was used. The Maximum shift algorithm demonstrated the best performance in all databases when a long pattern length was used. The two-way algorithm was the worst for both short and long patterns, whereas the Atheer showed the third best performance in most of the databases (Figures 8 and 9). In terms of the number of characters compared, the Atheer algorithm showed the best performance in all databases (except the DNA database) when the short pattern was used. The Atheer algorithm demonstrated the second best performance in all databases (except the XML database). The two-way algorithm was the worst when both short and long pattern lengths were used (Figures 10 and 11).

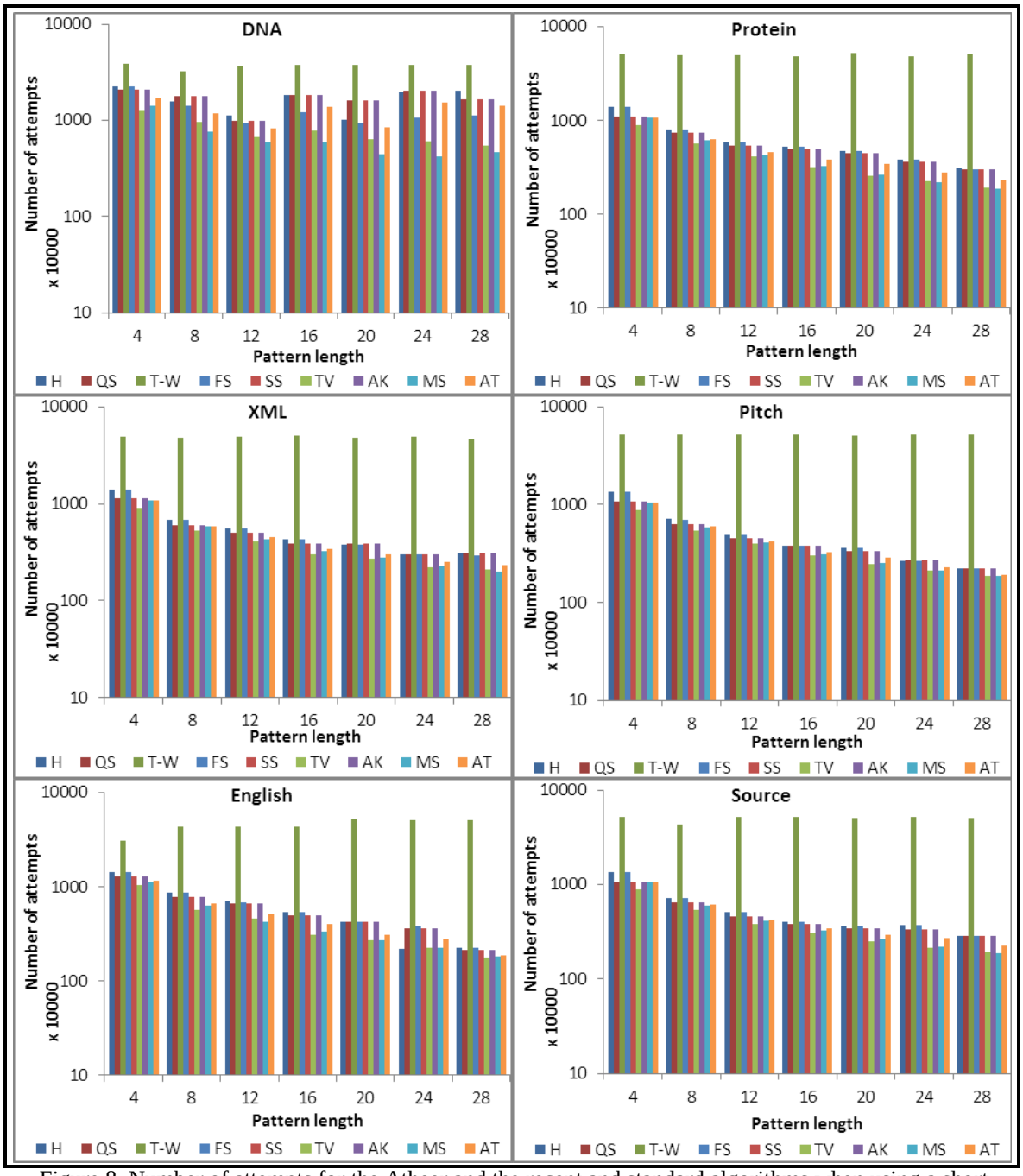

Figure 8. Number of attempts for the Atheer and the recent and standard algorithms when using a short pattern length and a database size of 50MB

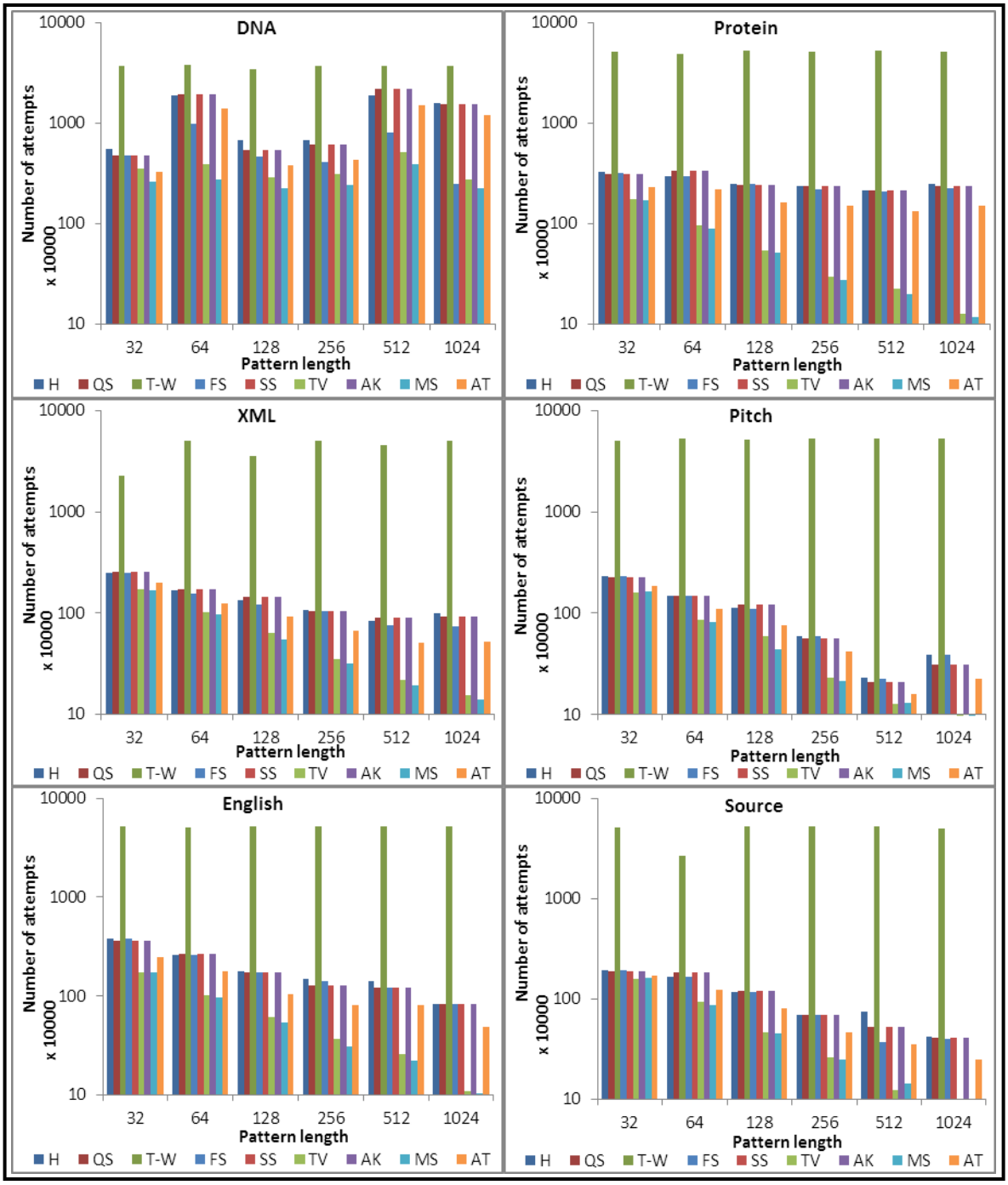

Figure 9. Number of attempts for the Atheer and the recent and standard algorithms when using a long pattern length and a database size of 50MB

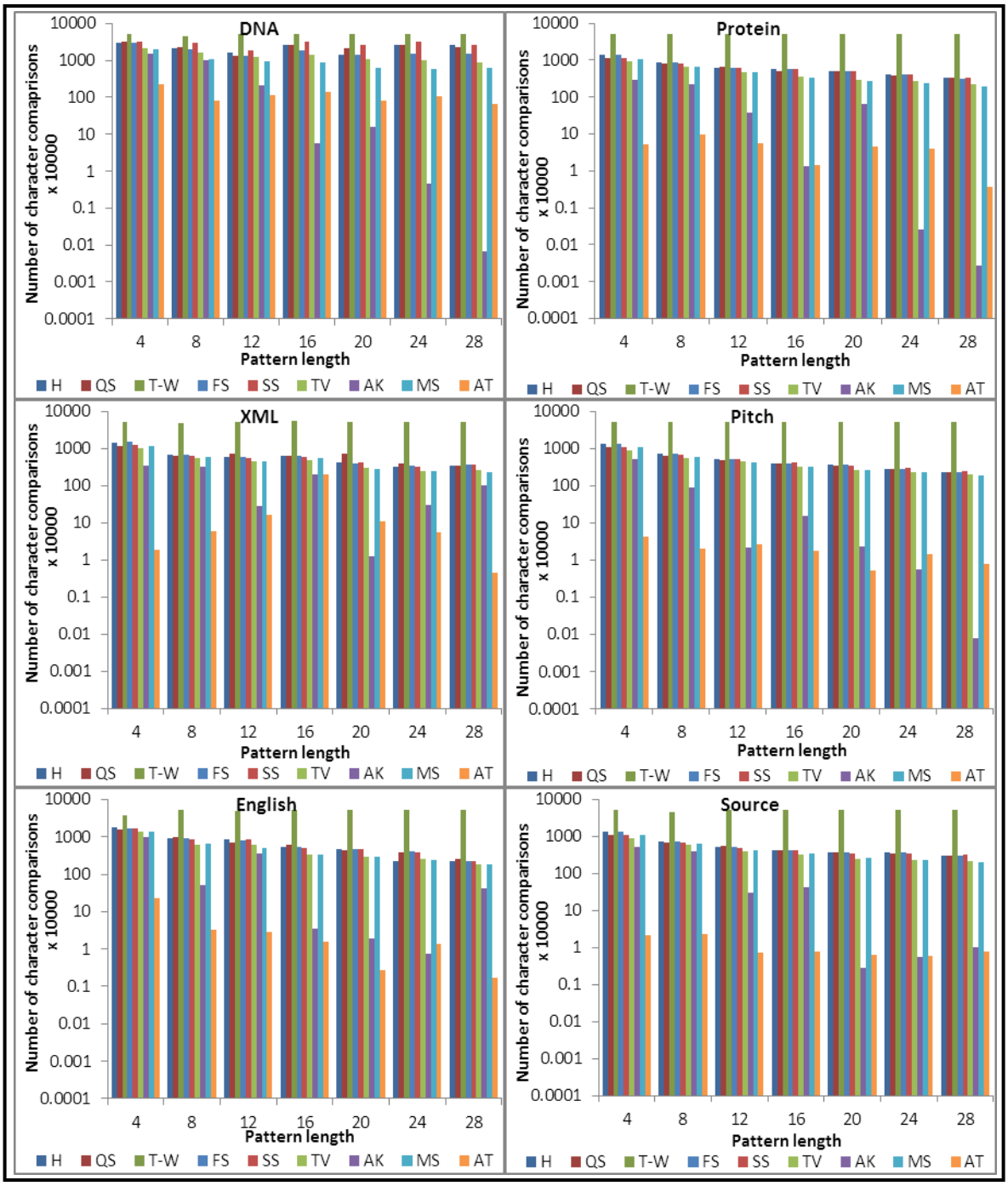

Figure 10. Number of characters compared for the Atheer and the recent and standard algorithms when using a short pattern length and a database size of 50MB

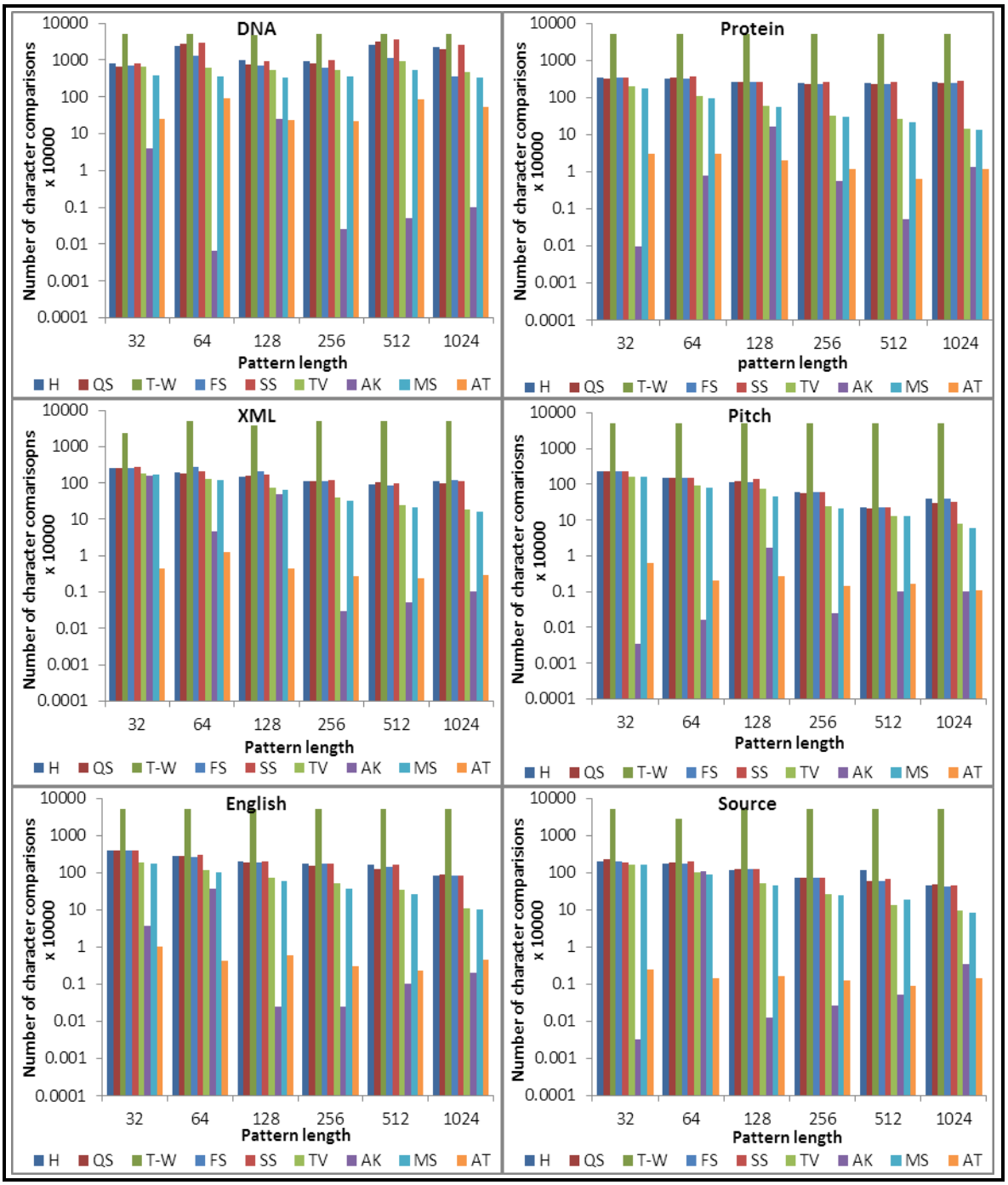

Figure 11. Number of characters compared for the Atheer and the recent and standard algorithms when using a long pattern length and a database size of 50MB

5. Discussion and analysis

The Atheer algorithm and the original algorithms were compared to highlight the excellent properties that can be obtained from the original algorithms. The Atheer algorithm was evaluated based on its best performance compared with the performance of the original algorithms (Faro and Lecroq, 2010).

5.1 Evaluation of the Atheer algorithm compared with the original algorithms

The results of the Atheer algorithm and the original algorithms were compared based on the number of comparison attempts and the number of characters compared using short and long pattern lengths, different data types, and a data size of 50 MB.

Table 2. Evaluation of the results of the Atheer algorithm and the original algorithms

Algorithms	Data size (50MB)	
	Short	Long
Best performance in number of attempts		
Raita	Third	Third
Karp-Rabin	Fourth	Fourth
Smith	First	First
Atheer	First	First
Best performance in number of character comparisons		
Raita	Fourth	Fourth
Karp-Rabin	First in all short pattern lengths (but second when used length equal 4)	First
Smith	Third	Third
Atheer	Second in all short pattern lengths (but first when used length equal 4)	Second

The Atheer algorithm obtained the best results in terms of the number of comparison attempts because its shifting depended on the good shifting process of the Smith algorithm. Therefore, both algorithms showed the lowest number of comparison attempts compared with the other original algorithms. The results were similar for the Karp–Rabin algorithm in terms of the number of characters compared. The Atheer followed the Karp–Rabin algorithms in showing the best results for the number of characters compared because of the hash function, which facilitates the comparison of characters in patterns and texts (Deighton, 2012). The Atheer obtained the second lowest number of characters compared in all pattern lengths, except in a very short pattern length such as 4. This result gave the first ranks to Atheer algorithm because the Karp–Rabin algorithm obtained high characters when very short pattern used but failed to do so with long pattern (Abdulrozaq, 2009). As the searching technique of the Atheer algorithm depends on a combination of the modified Raita technique and Karp–Rabin algorithm, it was able to obtain the best results when the very short pattern length was used (Table 2).

The Pitch database showed the best results compared with the other databases when the Atheer algorithm was utilized. Such results can be attributed to Karp–Rabin hash and the bmBc found in the Atheer algorithm, which can be considered as efficient functions when used with the Pitch database (Nidadavolu, 2008). Therefore, the Atheer algorithm showed the lowest number of attempts with the Pitch database. By using the hash function in large alphabet databases, the Atheer algorithm produced large hash values, which reduced the possibility of comparing the characters. The Source code database obtained the minimum number of characters compared. The DNA database obtained the highest number of attempts and character comparisons because it has a small alphabet size, which increased the possibility of comparing characters and reduced the long shifting process (Abdulrozaq, 2009) (Table 3).

Table 3. Performance of the Atheer algorithm in different types of databases

Performance	Database		
	Data size	50 MB	
	Pattern length	Short	Long
Attempts	Best	Pitch	Pitch
	Worst	DNA	DNA
Character comparisons	Best	Source	Source
	Worst	DNA	DNA

5.2 Evaluation of the Atheer algorithm compared with recent and standard algorithms

The Atheer algorithm was compared with the recent and standard algorithms according to the number of comparison attempts and the number of characters compared using short and long pattern lengths, different types of databases, and a data size of 50 MB. These standard and recent algorithms are Horspool, Quick search, Two-way, Fast search, SSABS, TVSBS, AKRAM, and Maximum shift.

Table 4. Comparison of the results of the Atheer versus those of recent and standard algorithms

Algorithms	Data size (50MB)	
	Short	Long
Number of attempts		
Best algorithm	TVSBS (most databases)	Maximum shift (all databases)
Worst algorithm	Two-way (all databases)	Two-way (all databases)
Number of character comparisons		
Best algorithm	Atheer (most databases)	AKRAM (most databases)
Worst algorithm	Two-way (all databases)	Two-way (all databases)

The TVSBS algorithm obtained the best number of attempts when the short pattern length was used because it depended on the bmBc function, which is considered as one of the useful functions in the shifting process (Thathoo et al., 2006). The Maximum shift algorithm obtained the best number of attempts when the long pattern length was used because it depended on the efficient function (ztBc) in long shifting (Kadhim, 2012). The Atheer algorithm showed the best number of characters compared when the short pattern was used because when a match was obtained for the three characters in the first step and a mismatch occurred in the next step, then the loss occurred only for the three characters. For the AKRAM algorithm, when a match was found in the suffix, which involved a non-specific number of characters, and a mismatch occurred in the next step, then the loss became equal

to the number of characters in the suffix (Abdulrozaq, 2009). Thus, the number of characters compared in the Atheer was lower than that in the AKRAM algorithm when short patterns were used. The number of characters compared in the AKRAM algorithm was lower than that in the Atheer algorithm because the suffix in the AKRAM algorithm obtained high accurate hash values, which reduced the possibility of mismatching (Abdulrozaq, 2009). The two-way algorithm obtained the highest results in terms of the number of attempts and characters compared because it depended on the factorization technique, which sometimes produces small suffixes that reduce the long shifting process and increase the number of characters compared (Charras and Lecroq, 2004) (Table 4).

Table 5. Positions of the Atheer algorithm in different types of databases

Databases	Position of Atheer algorithm	
	Pattern length	Data size 50 MB
Attempts	Short	Third in all databases
	Long	Third in all databases (but fourth in DNA)
Character comparisons	Short	First in all databases (but second in DNA)
	Long	Second in most databases (but first in XML)

For the number of comparison attempts, the Atheer algorithm ranked third in most of the databases with different sizes when short and long patterns were used. For the number of characters compared, the Atheer algorithm ranked first when short patterns were used in most of the databases with different sizes; it also ranked first in some of the databases when long patterns were used (Table 5).

6. Conclusion

The Atheer algorithm is a new hybrid string matching algorithm that is integrated with the advantages of the Karp–Rabin, Raita, and Smith algorithms. The Atheer algorithm performed better than the original algorithms did. This hybrid algorithm showed the lowest number of comparison attempts compared with the original algorithms, and it achieved the lowest number of characters compared when very short pattern lengths, such as 4, were used. The Atheer algorithm ranked the second best algorithm, following the Karp–Rabin algorithm, when short and long patterns were used. The Atheer algorithm also ranked the third best algorithm in terms of the number of comparison attempts, followed by the TVSBS and Maximum shift algorithms, when short and long patterns were used. The new algorithm also obtained the best number of characters compared when short patterns were used and the second best after the AKRAM algorithm when long patterns were used. The best and worst databases in terms of the number of comparison attempts for the Atheer algorithm were the Pitch and DNA databases, respectively. The best and worst databases in terms of the number of characters compared were the Source and DNA databases, respectively.

Acknowledgement:

This Research reported here is pursued under the Exploratory Research Grant Scheme (ERGS) by Ministry of Higher Education (Malaysia) and Universiti Sains Malaysia [203/PKOMP /6730074].

Corresponding Author:
Atheer Akram AbdulRazzaq
Department of Parallel and Distributed Processing
School of Computer Sciences, Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia
E-mail: athproof@yahoo.com

References

1. Faro S, Lecroq T. The Exact Online String Matching Problem: a Review of the Most Recent Results. ACM computing survey, 2013; 45(2): 1-42.
2. Bhukya R, Somayajulu D. Index Based Multiple Pattern Matching Algorithm Using DNA Sequence and Pattern Count. International Journal of Information Technology and Knowledge Management, 2011; 4(2)**:**431- 441.
3. Hasan AA, Rashid NA. Hybrid Exact String Matching Algorithm for Intrusion Detection System, Proceedings of the Second International Conference on Communications and Information Technology (ICCIT 2012), Al-Madinah Al-Munawwarah, Saudi Arabia, 2012;181-185.
4. Lu CW. String Matching Algorithms Based upon the Uniqueness Property and an Approximate String Matching Algorithm. Based upon the Candidate Master thesis, Department of Computer Science and Information Engineering, National Chi Nan University, 2008.
5. Klaib AF, Osborne H. A New String Matching Algorithm for Searching Biological Sequences. International Conference on Information and Communication Systems, ICICS, Amman, Jordan, 2009; 75-80.
6. Al-mazroi AA, Abdul Rashid N. (2011). A Fast Hybrid Algorithm for the Exact String Matching Problem, American Journal of Engineering and Applied Sciences, 2011; 4(1): 102-107.

7. Michailidis PD, Margaritis KG. On-line approximate string Searching algorithms: survey and experimental results. International Journal of Computer and Mathematic, 2002; 79(8): 867–888.
8. Zubair M, Wahab F, Hussain I, Zaffar J. Improved Text Scanning Approach for Exact String matching. International Conference on Information and Emerging Technologies. (ICIET), 2010; 1-5.
9. Hassan AA. Mixed Heuristic Algorithm for Intelligent String Matching for Information Retrieval. Proceedings of the Sixth International Conference on Computational Intelligence and Multimedia Applications. (IEEE), 2005; 11-16.
10. Cao F. Fast String Matching Algorithm and its Application in DNA Sequence Search. Master Thesis, School of Computer Science, Wayne State University, 2004.
11. Faro S, Lecroq T. The Exact String Matching Problem: a Comprehensive Experimental Evaluation. Data Structures and Algorithms. Report arXiv: 1012.254, 2010; 7: 1-22.
12. AbdulRazzaq AA, Abdul Rashid N, Ali ANB. Fast Hybrid String Matching Algorithm. International Journal of Digital Content Technology and its Applications (JDCTA), 2013a; 7(10): 62-71.
13. Karp RM, Rabin MO. Efficient Randomized Pattern-Matching Algorithms. IBM Journal of Research and Development, 1987; 31: 249–260.
14. Abdulrazzaq AA, Abdul Rashid N, Hamdani HBY, Ghadban RM, Mahmood AW. Influenced Factors on Computation Among Quick Search, Two-Way and Karp-Rabin Algorithms, Proceeding of the 3rd International Conference on Informatics and Technology (Informatics '09), Kuala Lumpur, 2009; 81-87.
15. Raita T. Tuning the Boyer-Moore-Horspool String Searching Algorithm. Software - Practice and Experience (SPE), 1992; 22(10):879–884.
16. Charras C, Lecroq T. Handbook of Exact String Matching algorithms. King's College Publications, France, 2004.
17. AbdulRazzaq AA, Abdul Rashid N, Hasan AA, Abu-Hashem MA. The exact string matching algorithms efficiency review. Global Journal on Technology (GJT), 2013b; 04: 576-589.
18. Smith PD. Experiments with a Very Fast Substring Search Algorithm. Software Practice and Experience, 1991; 21:1065–1074.
19. Karkkainen J, Joong CN. Faster Filters for Approximate String Matching Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX, USA, 2006; 84-90.
20. Chew E, Chen YC. Mapping Midi to the Spiral Array: Disambiguating Pitch Spellings. Operations Research/Computer Science Interfaces Series, 2003; 21:259-275.
21. Ferragina P, Fischer J. Suffix Arrays on Words. Springer-Verlag. Lecture Notes in Computer Science, 2007; 4580: 328–339.
22. Huang Y, Ping L, Pan X, Cai G. A Fast Exact Pattern Matching Algorithm for Biological Sequences. International Conference on Biomedical Engineering and Informatics, (BMEI) College of Computer Science and Technology, Zhejiang University, China2008; 8-12.
23. Cai G, Nie X, Huang Y. A Fast Hybrid Pattern Matching Algorithm for Biological Sequences. Proceedings of 2nd International Conference on Biomedical Engineering and Informatics, (BMEI '09), 2009; 1-5.
24. Deighton RA. Using Rabin-Karp fingerprints and LevelDB for faster searches, University of Ontario Institute of Technology (UOIT), Master thesis, School of Computer Science, Canada 2012.
25. Abdulrozaq AA. Fast Hybrid String Matching Algorithm Using Message Passing Programming Model. Master thesis, School of Computer Science, University Science Malaysia, 2009.
26. Nidadavolu R. Content-based Retrieval of Music Using Monophonic Queries on a Database of nd Polyphonic, MIDI Information. ProQuest information and learning company. United States, 2008; 21-22.
27. Thathoo R, Virmani A, Lakshmi SS, Balakrishnan N, Sekar K. TVSBS: A fast exact pattern matching algorithm for biological sequences. Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India. Current Science, 2006; 91: 47–53.
28. Kadhim HA. New Sequential and Gpu-Based Hybrid String Matching Algorithms. Master thesis, Computer Science School, University Science Malaysia, 2012.

6/25/2014