Quick-Skip Search Hybrid Algorithm For The Exact String Matching Problem

Asst. Lecturer Mustafa Abdul Sahib Naser

Software Eng. Department
Al-Mansour University College

Abstract:

The string matching problem occupies a corner stone in many computer science fields because of the fundamental role it plays in various computer applications. Thus, several string matching algorithms have been produced and applied in most operating systems, information retrieval, editors, internet searching engines, firewall interception and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. Several important factors are considered during the matching process such as number of character comparisons, number of attempts and the consumed time. This research proposes a hybrid exact string matching algorithm by combining the good properties of the Quick Search and the Skip Search algorithms to demonstrate and devise a better method to solve the string matching problem with higher speed and lower cost. The hybrid algorithm was tested using different types of standard data. The hybrid algorithm provides efficient results and reliability compared with the original algorithms in terms of number of character comparisons and number of attempts when the hybrid algorithm applied with different pattern lengths. Additionally, the hybrid algorithm produced better quality in performance through providing less time complexity for the worst and best cases comparing with other hybrid algorithms.

Keywords: character comparisons, amino acids search, exact pattern matching.

1. Introduction

String matching is used to check the similarities of strings. To solve the string matching problem it is necessary to find an algorithm which can locate the similarities of strings. The string matching procedure is an algorithm which compares a short string called pattern with a long string called text, its function is to check whether this pattern is a substring of the text. The procedure outputs location when a pattern occurs in the text and produces a mismatched signal when no pattern occurs in the text. In many fields, such as computer science, computer engineering, bio-science, lexical analysis, database query and so on, string matching processing is essential and therefore applied frequently [1].

A string matching problem can be defined as finding one or more occurrence of a given pattern string *P* of length *m* in a text string *T* of length *n*, which are built over a finite alphabet set *Σ* of size *σ*.

Definition 1: An alphabet *Σ* is a set of characters. The size of the alphabet is denoted by *σ* and represented by an integer number.

Definition 2: A string is a sequence of characters drawn from an alphabet. The input of the string matching algorithm are two strings, which are the pattern string $P = p_0 p_1 \dots p_{m-1}$ and the text string $T = t_0 t_1 \dots t_{n-1}$ where n ≥ m.

Generally, string matching algorithms scan the text with the aid of the sliding window mechanism. This mechanism involves opening a window on the text of which its size is equal to the pattern length *m*. Then it is followed by a comparison between the characters of the window and the characters of the pattern. This specific work of character comparison is called an attempt. After matching or mismatching all of the pattern characters with the window characters, the window is shifted along the text according to the heuristics of each algorithm [2].

Definition 3: A shift is defined as a safe skip to the number of characters without missing any occurrence of the pattern in the text [3].

Most of the exact string matching algorithms pre-process the pattern before searching the text. The purpose of the pre-processing phase is to maximize the length of the shift during the searching phase and that happens by collecting information about the pattern before starting the search of the pattern in the text. The searching phase involves different approaches for scanning the text to find the pattern occurrences in the text [4].

Development of the algorithms is considered a critical component in solving the problems when using the computer. The consumed time, performance, deficiency and cost are considered important factors in developing the algorithms. Many studies focus on the string matching problem. The hybrid algorithms are considered an example of such studies that deal with getting benefits from the original algorithms and overcome their weaknesses. Quick Search and Skip Search string matching algorithms are considered in this study, and these algorithms differ in their technique, performance, efficiency and usage.

The Quick Search is an efficient algorithm when using large alphabets with a short pattern during the text search [4, 5], but show less efficient behavior for small alphabets with a long pattern. On the other hand, the Skip Search algorithm [6]shows an efficient behavior when using small alphabets with a long pattern. Based on the reverse behavior of the two existing algorithms which deals with different alphabet types and different pattern lengths, along with the long consumed time wasted in searching big sized data, the important question that needs to be answered is "How to overcome the performance weaknesses of the two existing algorithms by proposing a hybrid algorithm which takes advantage of the positive characteristics of both algorithms to solve the string matching problem efficiently in any alphabet type and any pattern length?".

The rest of the paper is organized as follows. Section 2 gives the review of several efficient algorithms. Section 3 describes the proposed hybrid algorithm in detail. Section 4 analyses of the proposed hybrid algorithm are discussed. In Section 5, the experiment results of comparisons between the proposed algorithm and the original algorithms are given. And Section 6 is the conclusion.

I. PREVIOUS WORKS

The character comparison between the pattern and the text can be performed in different orders[2].This section classified the previous original string matching algorithm according to the direction of scanning the window and then discussed some of the previous hybrid algorithms.

A. From Left to Right

Brute Force (BF)[2] is the first string matching algorithm scans the character of the window from left to right and shifts the window exactly one position to the right after a mismatch or a complete match. The Knuth-Morris-Pratt (KMP)[7] algorithm is an improvement of the Brute Force (BF) algorithm, which uses a shift function based on the notion of the prefixes of the pattern and it is considered the first linear string matching algorithm. Skip Search and KMP Skip Search algorithms[6] behave like Knuth-Morris-Pratt algorithm by performing the characters of the window from left to right while the algorithms use buckets to determine the starting positions of the window in the text.The work of many algorithms depends on automaton theory with the Knuth-Morris-Prattconcepts.Search with an Automaton algorithm and Forward DAWG Matching (FDM) algorithm[8] work with the concept of the Knuth-Morris-Pratt (KMP) algorithm by performing the character comparisons from left to right. Search with an Automaton algorithm use the minimal Deterministic Finite Automaton (DFA), while Forward DAWG Matching algorithm uses the suffix automaton.Some of the algorithms use the nondeterministic form of the automata.Shift-Or (SO)[9] algorithm uses bit-wise operations for its work, while the algorithm performs character comparisons from left to right in the pattern and involves keeping a set of all the prefixes of the pattern that match a suffix of the text.

B. From Right to Left

The Boyer-Moore (BM)[10] algorithm is considered as one of the most efficient string matching algorithms which scan the characters of the window from right to left. There are many variants of Boyer-Moore algorithm which are widely recognized and used in various string matching applications.Like the Boyer-Moore type algorithms there are many algorithm works withthe automaton theory. The Reverse Factor algorithm[11] scans the characters of the window from right to left by calculating the smallest suffix automaton in the deterministic form of the reverse pattern. In the nondeterministic form of the automata, Backward Nondeterministic DAWG Matching (BNDM)[12] algorithm uses the suffix automaton of the reverse pattern in nondeterministic form which is simulated by using bit-parallelism.

C. In Any Order

Karp Rabin (KR)[13] algorithm uses the hashing methodology for string searching. The algorithm provides a simple and efficient method of avoiding quadratic number of character comparisons in most practical situations.

The aim of a good algorithm is to minimize the work done during each attempt and to maximize the length of the shifts to reduce the number of character comparisons through each attempt. Some of the algorithms deal with combining more than one algorithm to get an efficient advantage of the positive properties of these algorithms. This type of algorithms is called hybrid algorithms.

The SSABS algorithm [14] blends the advantages of Quick Search and Raita string matching algorithms. The authors proposed a fixed order of character comparisons between the window and the pattern during each attempt while the shifting of the window, after a complete match or a mismatch, depends on the Quick Search bad character function. Like Raita algorithm, SSABS algorithm compares the rightmost character of the window and the pattern at first and in the case of finding a match, the algorithm compares the leftmost character of the window and the pattern and also when finding a match, the remaining characters are compared from right to left. In case of a mismatch in any of the existing comparisons, the algorithm does not compare the remaining characters and shifts the window by the Quick Search bad character function.

TVSBS algorithm [15] is a combination of Berry–Ravindran and SSABS algorithms. The resulting hybrid algorithm is efficient for applications related to biological sequence search. In the pre-processing phase, the TVSBS algorithm calculates the Berry-Ravindran bad character function with suitable modifications. It stores the bad character shift values in the one-dimensional array instead of a two-dimensional array to reduce the accessing time during the searching phase. The searching phase for this hybrid algorithm is the same as the SSABS algorithm. The procedure of the TVSBS algorithm presents goodness in application related to exact string matching in biological sequence database.

BRFS algorithm [16] is the result of combining the Fast Search (FS) and Berry-Ravindran (BR) string matching algorithms. The pre-processing phase of this hybrid algorithm consists of computing the Boyer-Moore's good suffix function and Berry-Ravindran's bad character function. The searching phase procedure is the same as the Fast Search algorithm which performs character comparisons from right to left until a complete match or a mismatch occurs. The BRFS algorithm has better performance for small alphabets with a long pattern. It is therefore suitable for the application related to biological sequence search.

All the mentioned hybrid algorithms are the result of hybridizing two or more algorithms. They have advantage characteristics in the performance over the original algorithms. This performance makes the hybrid algorithm to show robustness and better behavior in different applications by increasing the shift value and decreasing the number of character comparison and the time required in the search procedure.

II. THE PROPOSED ALGORITHM

This section discusses the proposed hybrid solution to combine the Quick Search and the Skip Search algorithms. Like the two existing algorithms, the efficiency of the proposed hybrid algorithm lies in two phases which are the pre-processing phase and the searching phase. The characters in the pattern are pre-processed in the pre-processing phase and this information is used in the searching phase in order to reduce the total number of character comparisons as well as number of attempts.

A. Pre-processing Phase

The pre-processing phase for the proposed hybrid algorithm includes the process of building the pre-processing phase from both original algorithms. The pre-processing phase for the hybrid algorithm is constructed by building the Quick Search bad character table and the Skip Search buckets.

The reason for using unincorporated method to construct the pre-processing phase for the proposed hybrid algorithm from the two original algorithms is due to the different techniques of constructing the Quick Search bad character table(qsBc) and the Skip Search buckets. The Quick Search bad character table contains the rightmost location for each alphabet in the pattern, while the Skip Search buckets contain the leftmost location for all characters in the pattern.

The information getting from the pre-processing phase is used in the searching phase in order to reduce the total number of character comparisons as well as the number of attempts.The pre-processing phase goes hand-in-hand with the searching phase to improve the overall efficiency of the algorithm by calculating larger shift values.

B. Searching phase

The techniques in this phase depend on the searching phase of the original algorithms using different orders with modification during the matching operation. In general, the searching phase of the hybrid algorithm will be arranged in several stages. These stages clarify the work of the hybrid algorithm during the matching operation.

Stage 1:atthis stage, the algorithm examines the starting search point Swhich has a position T_j in the text, whereas j is equal to the pattern length m. The algorithm aligns the character of this position and the pattern with the corresponding position of this character in the bucket. The benefit of this operation is that when the character in position T_j does not occur in the pattern, the algorithm continues shifting the pattern to the next T_j position in the text. In order to avoid many character comparisons, this operation avoids aligning the leftmost character of the pattern and the window at the beginning of the searching phase. Furthermore, the algorithm ensures that there is no possibility of a matching occurring during the process of shifting the pattern to align the next T_j position.

Stage 2:this stage follows the chosen starting search point in stage 1. At this stage, comparisons occur between the characters of the pattern and the window. The first comparisons of the characters start from the leftmost character of the pattern with the corresponding position of this character in the window. If a complete match or a mismatch between the characters happens, the algorithm moves to the next stage.

Stage 3:atthis stage, the algorithm calculates the shift value of the Skip Search and the Quick Search respectively. The Skip Search shift value of the hybrid algorithm is calculated differently depending on two situations. The first situation is when the character in the pattern (which matches the corresponding position of T_j in the text) occurs in the last position of the bucket. The shift value of this situation is calculated by the following equation after discriminating the first bucket position of the character which occurs in the next T_j position of the text which is considered the next start search point.

Skip shift = m + the current position of T_j (from the bucket) – the next position of T_j ….. (1)

The second situation is when the character in the pattern (which matches the corresponding position of T_j in the text) does not occur in the last position of the bucket. The shift value of this situation is calculated by subtracting the next position value from the current position value of this character in the bucket.

The Quick Search shift value of the hybrid algorithm is assigned for a character immediately next to the window. This depends on the value of the rightmost occurrence of that character in the pattern which is recorded in the Quick Search bad character table.

After calculating the Skip Search and the Quick Search shift values, the algorithm examines the bigger shift. If the Skip Search shift is bigger, then the algorithm depends on which Skip Search situation should be applied as shown in Figure 1. If the shift amount of the Skip Search is equal to the Quick Search shift, then the algorithm depends on the Skip Search shift and moves to *Stage 2*. Otherwise, the algorithm moves into the next stage.

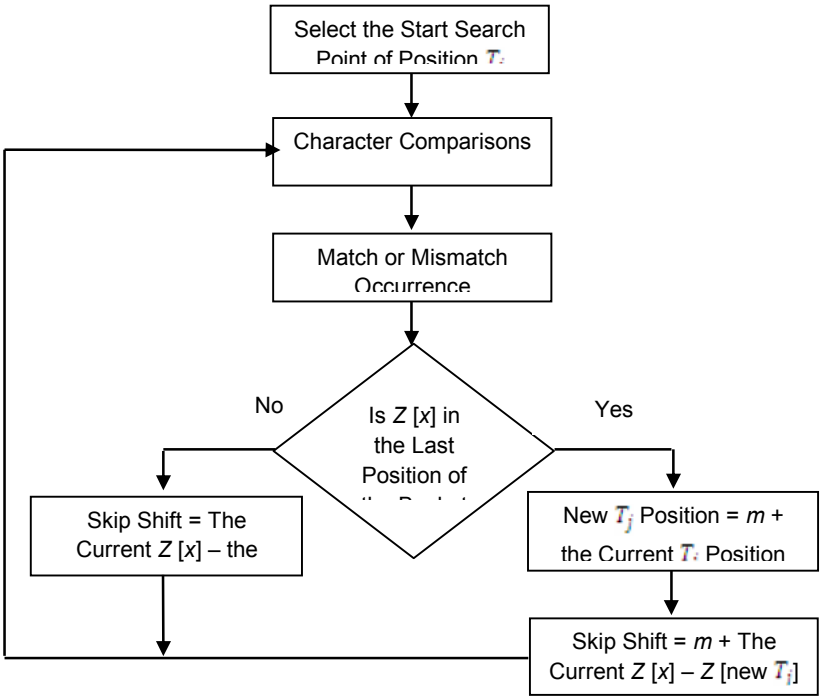

Fig.1. Skip Search Shift in the Hybrid Algorithm

Stage 4: this stage is applied if the hybrid algorithm depends on the Quick Search shift. The operation of the Quick Search shift in the hybrid algorithm depends on two situations. The first situation is when the value of the character immediately next to the window is less than or equal to the pattern length m. In this situation, the current position of T_j in the text moves in order to become equivalent to the character's position immediately next to the window which is considered to be the new start search point and the algorithm directly moves to Stage 2 as shown in the following equation.

If (Quick Search Shift > Skip Search Shift) and (Quick Search Shift ≤ m)

Then

Current Position of T_j = Position Immediately Next to the Window….. (2)

The second situation is when the value of the character immediately next to the window is bigger than the pattern length *m*. In this situation, the current position of T_j in the text moves in order to become equivalent to the character position immediately next to the window plus the pattern length *m*. This position is considered to be the new start search point if the character in this position occurs in the pattern. Otherwise, the algorithm continues shifting the

pattern to the next possible start search point and also the algorithm directly moves to *Stage 2* as shown in the following equation.

If (Quick Search Shift > Skip Search Shift) and (Quick Search Shift > m)

Then

Current Position of T_j = Position Immediately Next to the Window + m….. (3)

Figure 2 shows the function of the Quick Search shift during the searching phase of the hybrid algorithm. All the stages of the searching phase are repeated until the window is positioned beyond *n* – *m* + 1.

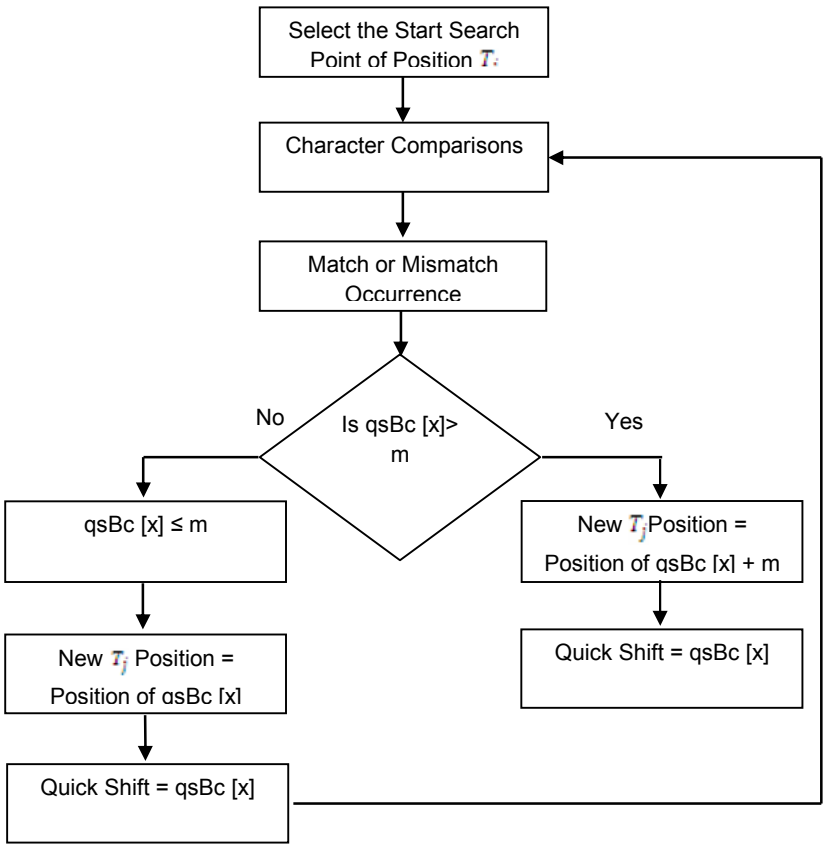

Fig.2. Quick Search Shift in the Hybrid Algorithm

III. ANALYSIS OF THE PROPOSED ALGORITHM

The pre-processing phase of the proposed hybrid algorithm is constructed by building the pre-processing phase of both original algorithms used in the hybridization method. Since the two original algorithms have the same pre-processing time complexity which is O(m+σ), the pre-processing time complexity of the hybrid algorithm is assigned for combining the time complexity of both original algorithms and hence equals to O(2(m+σ)).

During the searching phase, the key factors defining the average time complexity are the possibility of each individual character occurring in the text and the alphabet size.

Because both these factors are highly indiscriminate and the lack of any reliable prediction mechanism, this study admits that the average time complexity cannot be exactly defined[15]. According to that, the searching phase time complexity for the proposed algorithm assigns for the worst and best cases.

***Lemma 3.1*: The time complexity is O(*nm*) in the worst case.**

Proof: The worst case algorithm occurs when all the characters of the pattern match with the characters of the text at each attempt. This case can be realized when all the characters in the pattern are the same as those in the text. During this situation, the hybrid algorithm depends on the shift provided by the skip shift only. According to that, every character in the text is matched no more than *m* times and the total character comparisons for *n* characters of the text cannot be more than (*nm*), whereas the shift in this case is equal to one and hence the time complexity is O(*nm*).

Example 1:

Text =”A AAAAAAAAAAAAAAAAAA”

Pattern =”A AAAA”

The text length (n) = 20.

The pattern length (m) = 5.

The alphabet set (Σ) = (A) of size (σ) = 1.

Lemma 3.2: The time complexity is O($^{n}/_{m}$) in the best case.

Proof:The best case complexity of the proposed hybrid algorithm occurs when the characters of the pattern are totally not matched with any character in the text at any attempt. This case can be realized when all the characters in the pattern are completely different from those in the text. In this case and according to the hybrid algorithm behavior, the algorithm will check the *m*-th text positions to delimit the possible starting search point *S* in the text. Since there is no match at all, the algorithm will provide $^{n}/_{m}$main iterations during the searching phase without any charactercomparisons and attempts until the pointer reaches to the end of the text and hence the time complexity is O($^{n}/_{m}$).

Example 1:

Text =”A AAAAAAAAAAAAAAAAAA”

Pattern =”B BBBB”

The text length (n) = 20.

The pattern length (m) = 5.

The alphabet set (Σ) = (A, B) of size (σ) = 2.

As a comparison to examine the performance of the proposed hybrid algorithm, the worst and best time complexity is compared with two other hybrid algorithms stated in the literature. Also, the comparison includes the pre-processing time complexity of each algorithm as shown in Table 1.

Table 1: Comparisons of Hybrid Algorithms Complexity

Algorithms	Pre-processing Time Complexity	Searching Time Complexity	
		Worst Case	Best Case
TVSBS Algorithm	O(σ+ *m*σ)	O(*m*(*n* – *m* + 1))	O(*n*/(*m* + 2))
BRFS Algorithm	O(*m*+σ^2)	O(*nm*)	O(*n*/(*m* + 2))
Quick-Skip Search Algorithm	O(2(*m*+σ))	O(*nm*)	O(*n*/*m*)

IV. EXPERIMENTAL EVALUATION

A standard benchmark data is used which illustrates the common uses of the string matching application. These types contain the DNA sequence, protein sequence and English text. The reasons of selecting these specific type of data is because they differ in the size of alphabets in order to examine various algorithm behaviors with various alphabet sizes, while the size of the data types used was 100 MB.

In order to analyze and discuss the actual behavior and to decrease the random variation for each algorithm, the running occurs in 5 times with different patterns for each length. The patterns lengths are: 4, 6, 8, 10, 20, 40, 60, 80 and 100 characters which are chosen randomly from words inside the text while five patterns were searched for each length and then take the average. The results of the proposed hybrid algorithm compared with the original algorithms in terms of number of character comparisons and number of attempts.

The working environment used in implementing the algorithms is a personal computer with 2.0 GHz Intel Core 2 Duo Processor, and 2 GB of RAM. The operating system used in this experiment is Microsoft Windows Vista service pack2, with Microsoft Visual C++ compiler.

1. Evaluating the Number of Character Comparisons

A. Average Running Times of DNA Sequence Data Type

Table 2: Average Number of Character Comparisons for All the Pattern Lengths of DNA Sequence. Alphabet size (σ) = 7.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	41374497	49366673	35223805
6	39748644	45175832	32964853
8	37363660	44876000	28787361
10	36441125	43300707	25834517
20	35614031	29206874	24974752
40	34708725	29633987	22010773
60	34378964	27758511	20889449
80	34215572	29451867	19771123
100	33181347	31352130	16372368

B. Average Running Times of Protein Sequence Data Type

Table 3: Average Number of Character Comparisons for All the Pattern Lengths of Protein Sequence. Alphabet size (σ) = 20.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	8607556	25545595	7406043
6	7887639	18889706	6365317
8	6959901	16089619	5760243
10	7418229	14185255	4829583
20	7458850	8947894	4356758
40	6635051	6899484	3953897
60	6795244	6151638	3724476
80	6694680	5644213	3533147
100	6689656	4985535	3518769

C. Average Running Times of English Text Data Type

Table 4: Average Number of Character Comparisons for All the Pattern Lengths of English Text. Alphabet size (σ) = 100.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	9590612	30457873	7519612
6	8888589	21272393	6242738
8	8729178	18065563	5495307
10	6832820	14841982	5380871
20	6728571	9519256	4690644
40	6877695	6448313	4014436
60	6831379	5401421	3443087
80	7164099	4451513	2779310
100	6909799	4117679	2626494

2. Analyzing Number of Character Comparisons

Based on the empirical results shown in table 2, 3, 4, it is clear that the DNA data type produces larger results for number of character comparisons compared with other data types especially when using short pattern lengths. This result is caused by the size of the alphabets used which are considered as a small alphabet size. This leads to producing less number of shifts during the searching operation which leads to a larger number of character comparisons. Furthermore, when a small sized alphabet is used it leads too many exact matching between the pattern and the window especially when using short pattern lengths and as a result the number of character comparisons tends to be greater than using large alphabet sizes. Also, it must be observed that for all algorithms, the number of character comparisons tends to decrease significantly as the pattern length increases.

This is because, the shift provided by the algorithms increases if the mismatch occurs, by that increasing the forward distance taken by the pattern. In all cases, it can be seen that the hybrid algorithm produces better results. The hybrid algorithm is highly efficient in terms of number of character comparisons than the original algorithms for short and long patterns respectively as well as when using different data types.

3. Evaluating the Number of Attempts

A. Average Running Times of DNA Sequence Data Type

Table 5: Average Number of Attempts for All the Pattern Lengths of DNA Sequence. Alphabet size (σ) = 7.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	25282517	35192710	22389453
6	25895761	32909817	20708859
8	25276133	30812265	18902520
10	25651039	30823717	17524454
20	24899958	22026198	18046927
40	25837273	21718409	15779217
60	25665193	20339958	15729218
80	26261647	23027420	14585324
100	25474945	24188623	12323918

B. Average Running Times of Protein Sequence Data Type

Table 6: Average Number of Attempts for All the Pattern Lengths of Protein Sequence. Alphabet size (σ) = 20.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	6492151	23281128	5683046
6	6338840	17250592	5095329
8	5802705	14438298	4947206
10	6257809	12864914	4368470
20	6535248	8386797	3900834
40	6234336	6422117	3712920
60	6288577	5751047	3443499
80	6194243	5320302	3308175
100	6287236	4646832	3291172

C. Average Running Times of English Text Data Type

Table 7: Average Number of Attempts for All the Pattern Lengths of English Text. Alphabet size (σ) = 100.

Pattern Length	Algorithm		
	Skip Search (SS)	Quick Search (QS)	Hybrid (QSS)
4	6969696	24335559	5647706
6	6734139	18474471	5177682
8	6597203	14951568	4818332
10	6139410	12654099	4725219
20	5851183	8403953	4300581
40	6134122	5852870	3694405
60	6367043	5041604	3130391
80	6482366	4082904	2623275
100	6189312	3866570	2443532

4. Analyzing Number of Attempts

Based on the empirical results shown in table 5, 6, 7, we can observe that for all data types the results for number of attempts provided by the Skip Search algorithm did not change significantly when the pattern length changed. Also, we can observe that the Skip Search algorithm produced less number of attempts than the Quick Search algorithm when short pattern lengths were used.

It should be noted that the number of attempts produced by the Quick Search algorithm decreases when the pattern lengths increases for all data types except when using DNA data type with long pattern lengths. In this situation, the algorithm shows unstable behavior and this is caused by the small size of alphabets used as well as the bad behavior for the Quick Search bad character table when small alphabets with long pattern lengths were used. However, the Quick Search algorithm provided less number of attempts than the Skip Search algorithm when long pattern lengths were used in all data types.

The obtain result experimentally demonstrated that the two original algorithms differ in behavior when using different alphabet sizes with different pattern lengths during the searching operation. Additionally, our experiments confirm that the hybrid algorithm outperform the two original algorithms in number of attempts when different alphabet sizes with different pattern lengths were used.

V. CONCLUSION

This paper aims to hybridize the Quick Search and Skip Search exact string matching algorithms. Based on the design presented in section three, the hybridization method produced an algorithm depending on the good properties of the original algorithms.

The performance of the proposed hybrid algorithm has shown improvement when compared with the original algorithms. The hybrid algorithm provided better results in number of character comparisons and number of attempts when searching different data types with different pattern lengths than the original algorithms. Therefore, it is feasible that this method can be used in applications related to exact pattern matching with any alphabet type.

REFRENCES

1. G. Navarro and M. Raffinot**,** ***Flexible Pattern Matching in Strings:*** *Practical On-line Search Algorithms for Texts and Biological Sequences*, Cambridge University Press 2002
2. C. Charras and T. Lecroq**,** ***Handbook of Exact String Matching Algorithms*****,** King's Collge Publications, 2004.
3. Y. Weinsberg, Tzur-David, S., Dolev, D. & Anker**, T. (2007) One Algorithm to Match Them All:** On a Generic NIPS Pattern Matching Algorithm. Workshop on High Performance Switching and Routing (HPSR2007). pp. 1-6., "One Algorithm to Match Them All," 2007, pp. 1-6.
4. T. Lecroq, **"Experimental Results On String Matching Algorithms,"** *Software Practice And Experience*, vol. 25, 1995, pp. 727-765.
5. A.F. Klaib, Zainol. Z., Ahamed, N. H., Ahmad, R. &Hussin, W.**, "Application of Exact String Matching Algorithms towards SMILES Representation of Chemical Structure,"** *International Journal of Computer and Information Science and Engineering*, 2007, pp. 235-239.
6. C. Charras, Lecroq, T. &Pehoushek, J. D**. , " A Very Fast String Matching Algorithm For Small Alphabets And Long Patterns,"** *Proceedings of the Ninth Annual Symposium on Combinatorial Pattern Matching, Lecture notes in computer science*, vol. 1448, 1998, pp. 55-64.
7. M.K.G. Michailidis. P. D. **, "On-line String Matching Algorithms: Survey and Experimental Results,"** *International Journal of Computer and Mathematic*, vol. 76, 2000, pp. 411-434.
8. N. Nadia**, "Minimal deterministic left-to-right pattern-matching automata,"** *SIGPLAN Not.*, vol. 33, no. 1, 1998, pp. 40-47**;**
9. G.G.H. Baeza-Yates. R. **, "A New Approach To Text Searching,"** *Communications of the Association for Computing Machinery (ACM 1994)*, vol. 35, 1992, pp. 74-82.
10. R.S.M. Boyer, J. S. **, "A Fast String Searching Algorithm,"** *Communications of the Association for Computing Machinery (ACM 1994)*, vol. 20, 1977, pp. 762-772.
11. C. Maxime, et al., **"Speeding Up Two String-Matching Algorithms,"** *Book Speeding Up Two String-Matching Algorithms*, Series Speeding Up Two String-Matching Algorithms, ed., Editor ed.^eds., Springer-Verlag, 1992, pp**.**
12. G. Navarro and M. Raffinot**, "A bit-parallel approach to suffix automata:** Fast extended string matching "*Springer*, vol. Volume 1448/1998, 1998, pp. 14-33; DOI 10.1007/BFb0030776.
13. R.M.R. Karp, M. O. **, "Efficient randomized pattern-matching algorithms,"** *IBM Journal of Research and Development*, vol. 31, 1987, pp. 249-260.
14. S.S. Sheik, Aggarwal, S. K., Poddar, A., Balakrishnan, N., Sekar, K**., " A Fast Pattern Matching Algorithm,"** *Journal of Chemical Information and Computer Sciences*, vol. 44, 2004, pp. 1251-1256.
15. R. Tathoo, Virmani, A., Lakshmi, S. S., Balakrishnan, N., Sekar, K**.."TVSBS: A Fast Exact Pattern Matching Algorithm for Biological Sequences,"** *Journal of Indian Academy of Sciences*, vol. 91, 2006, pp. 47-53.
16. H. Yong, et al.**, "A Fast Exact Pattern Matching Algorithm for Biological Sequences,"** *Proc. International Conference on BioMedical Engineering and Informatics, 2008. BMEI 2008.*, 2008, pp. 8-12.

خوارزمية (القفز- السريع) الهجينة لحل مشكلة البحث عن تطابق السلسة التام

م. م. مصطفى عبد الصاحب ناصر

كلية المنصور الجامعة

المستخلص

مشكلة البحث عن تطابق السلسلة تحتل حجر الزاوية في العديد من مجالات علوم الحاسبات بسبب الدور الأساسي الذي تلعبه في مجال تطبيقات الحاسوب المختلفة. لذالك, فقد تم اقتراح العديد من الخوارزميات لحل هذه المشكلة وتطبيقها في معظم نظم التشغيل واسترجاع المعلومات ومحركات البحث على الانترنيت وانظمة حماية الحاسبات والبحث عن سلسلة معينة من الاحماض الامينية في قواعد بيانات سلسلة الجينوم والبروتين. هناك عدة عوامل مهمة يجب مراعاتها خلال عملية البحث عن التطابق مثل عدد الرموز التي يجب مقارنتها وعدد مرات التطابق والوقت المستهلك في عملية البحث عن سلسلة معينة. هذا البحث يقدم خوارزمية تطابق هجينة من خلال الجمع بين الخصائص الجيدة لخوارزمية البحث السريع و خوارزمية القفز وأيجاد أفضل طريقة لحل مشكلة البحث عن التطابق بسرعة عالية وكلفة اقل. حيث تم اختبار الخوارزمية الهجينة باستخدام أنواع مختلفة من البيانات القياسية حيث ان الخوارزمية الهجينة وفرة نتائج كفونة وذات موثوقية عالية مقارنة مع الخوارزميات الأصلية من حيث توفير عدد اقل من المقارنات خلال عملية البحث وعدد اقل من محاولات التطابق عند تطبيق الخوارزمية الهجينة بأستخدام انماط مختلة الاطوال. بالإضافة إلى ذلك ، تتيح الخوارزمية الهجينة المقترحة نوعية أداء افضل من حيث التعقيد في أسوأ وافضل حالات المقارنة حينما تقارن مع خوارزميات هجينة اخرى.