# An algorithm to improve the performance of string matching

Abdallah A. Hlayel and Adnan Hnaif

*Journal of Information Science* 2014 40: 357 originally published online 14 January 2014
DOI: 10.1177/0165551513519039

Article

Journal of Information Science
2014, Vol. 40(3) 357–362
© The Author(s) 2014
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0165551513519039
jis.sagepub.com

# An algorithm to improve the performance of string matching

**Abdallah A. Hlayel**
Department of CIS, Faculty of Science and IT, Al-Zaytoonah University of Jordan, Jordan

**Adnan Hnaif**
Department of Computer Science, Faculty of Science and IT, Al-Zaytoonah University of Jordan, Jordan

**Abstract**
Approximate string matching algorithms are techniques used to find a pattern 'P' in a text 'T' partially or exactly. These techniques become very important in terms of performance and the accuracy of searching results. In this paper, we propose a general approach algorithm, called the Direct Matching Algorithm (DMA). The function of this algorithm is to perform direct access matching for the exact pattern or its similarities within a text depending on the location of a character in alphabetical order. We simulated the DMA in order to show its competence. The simulation result showed significant improvement in the exact string matching or similarity matching, and therefore extreme competence in the real applications.

**Keywords**
Approximate matching algorithms; direct matching algorithm; edit distance algorithm; pattern matching; string matching algorithms

## 1. Introduction

In many fields of computer science applications, such as network security, artificial intelligence and data retrieval, it is necessary to use one of the string matching algorithms, which find the occurrence of pattern 'P' in a text 'T', where 'T' is longer than 'P' [1, 2]. This matching is either exactly or partially matched with the pattern. Thus, string matching algorithms can be divided into two categories: exact string matching algorithms and approximate string matching algorithms. Exact string matching algorithms can be used to find all occurrences of 'P' in 'T', while approximate string matching algorithms are concerned with the similarity percentage between 'P' and 'T' [3, 4]. String matching algorithms deal with the concepts of identification and authentication in the field of network security [5]. Also, string matching can be used in sequence matching in large DNA databases [6]. Much research has been performed to enhance the performance of string matching algorithms [7, 8].

This paper introduces an algorithm called the Direct Matching Algorithm (DMA), which can be used to find either exact matching or approximate matching between two strings, so that the proposed algorithm consists of two steps: first, create a matrix called the matrix index, in order to arrange all of the characters of the text in the matrix index based on their positions; second, run the matching process to find all the occurrences of the pattern 'P' and the text 'T'.

## 2. Related work

In the following section, we indicate some of the best-known algorithms that are used in exact string matching algorithms and approximate matching algorithms.

**Corresponding author:**
Abdallah A. Hlayel, CIS Department, Faculty of Science and IT, Alzytoonah University of Jordan, Amman, Jordan.
Email: hlayel@zuj.edu.jo

**Table 1.** Dynamic programming algorithm to compute the ED between 'hand' and 'hard'.

| | | H | A | N | D |
|---|---|---|---|---|---|
| | **0** | 1 | 2 | 3 | 4 |
| H | 1 | **0** | 1 | 2 | 3 |
| A | 2 | 1 | **0** | 1 | 2 |
| R | 3 | 2 | 1 | **1** | 1 |
| D | 4 | 3 | 2 | 2 | **1** |

## 2.1. Exact string matching algorithms

The Boyer–Moore algorithm is one of the best-known pattern matching algorithms, and is considered to be very fast in practice. It was designed for the exact string matching of many strings against a single keyword [9]. The first heuristic phrase used is 'bad character shift'. Bad character shift starts a comparison from the right to the left and, if a character is seen that does not exist in the text to search for, then the search algorithm can be shift forwards to an 'M' character, where 'M' is the length of the pattern. The second heuristic phrase used in the Boyer–Moore algorithm is 'good suffix shift'. Good suffix shift starts a comparison from the right to the left and, if it matches, the algorithm checks the next character in the text with the next character in the pattern, until matching all the strings. In the case of mismatching, the Boyer–Moore algorithm looks for the next occurrence of a substring that was matched before.

## 2.2. Approximate matching algorithms

The edit distance algorithm is considered the best algorithm to be used to find the distance between two strings. The edit distance between two strings 's' and 't' is the minimum number of edit operations needed to convert the string 's' into 't'. Some algorithms have been proposed in order to calculate the Levenstine edit distance efficiently, such as the Dynamic Programming Algorithm, which is used to improve the efficiency of the problem solution. Table 1 depicts the use of the Dynamic Algorithm to compute the edit distance (ED) between the two strings ('HAND') and ('HARD').

As shown by Table 1, the path to the final result is shown by the bold entries. From above example, the time complexity for the algorithm is $O(|s||t|)$ in the worst and average cases, and the space complexity is only $O(\min(|s|,|t|))$ because, using column-wise processing, the previous column only must be stored in order to compute the new one, assuming that the number of rows in the matrix equals the length of the shorter string.

## 3. The proposed DMA and its implementation

This section will discuss the DMA. The DMA applies a different technique from the ED algorithm, to match the pattern 'P' in the text 'T'. This technique must create an alphabetical index matrix 'M', and then apply the matching process.

The proposed algorithm enhances the speed of matching between the pattern 'P' and the text 'T', through creating the alphabetical index matrix 'M', which re-arranges all of the character positions of the text 'T' in 'M' according to the corresponding characters. The index matrix runs only once, as long as no updates are available for text 'T' in case of matching more than one pattern. In addition, the DMA matching process deals with the values of the character position indices directly. The DMA is divided into three stages.

## 3.1. Preparation stage

In order to evaluate the matching process between two strings, the preparation stage will run first to create alphabetical index matrix 'M' and fill character positions of the text 'T' in the 'M' according to the corresponding characters.

## 3.2. Matching stage

The DMA can be used to find all the possible similarities between the pattern 'P' and the text 'T'. Consequently, the matching stage works as per follows steps:

(1) Read the pattern 'P', and then create the array list 'L', which will contain all the possible matching similarities between 'P' and 'T'.

(2) Read the first character of the pattern ‘P’ ($i = 1$), and check it with the corresponding character in ‘M’, where this character should have an index value that does not equal null (not null). Otherwise, read the second character of the pattern ‘P’ ($i = 2$) and so on.
(3) In case of matching index for first character, the DMA will take the index of the matching character and add it to the array list ‘L’ under the corresponding position.
(4) Read the next character of the pattern (‘P’ ($i + 1$)), and match it with the characters of ‘M’. If matching, then take its index, the value of which is greater than the index value of the current character ‘P’($i$), and also is close to it.
(5) Repeat step ‘4’ for the subsequent characters until reaches the end of the pattern ‘P’, or until an exact match is obtained.
(6) Repeat steps 2 to 5 with the next index value for the first matching character to obtain the next possible match of similarity array ‘L’ until reaching the last index for the first character, or until an exact match is obtained.

### *3.3. Sorting the similarity results in descending order*

In order to obtain the maximum percentages of matching or an approximate matching between the pattern ‘P’ and the text ‘T’, stage 3 works for this reason. From the array list ‘L’, all of the similarity results will be sorted in descending order. So, going down in ‘L’, the similarity percentage of matching will increase (the last possible matching has the maximum percentage of matching between the pattern ‘P’ and the text ‘T’).

## 4. Results and business problem

### *4.1. Problem*

Consider the problem of matching pattern P = ‘gcagagag’ in the text T = ‘gcatcgcagagagtatacagtacg’. First we apply the preparation stage, where the alphabetical index matrix ‘M’ is created as shown by Table 2.

This is a very important step for speeding up the matching process, because as we have said, this step only runs once, as long as no updates are available in the text ‘T’. Also, by using the DMA, we can access the character positions directly. Regarding the matching stage, Table 3 depicts step 1 of the proposed matching algorithm, which describes the process of reading the pattern ‘P’ and creating the array list ‘L’.

As shown by Table 3, the first character is ‘g’, and by reference to the index matrix ‘M’, it has the indices {1, 6, 9, 11, 13, 24} where the first index of the character ‘g’ is 1. Regarding the DMA step 3, we can add the index ‘g’ to the array list ‘L’ under the corresponding position (see Table 4).

After that, step 4 is applied to read the next character of the pattern ‘P’, which is ‘c’ and by reference to the index matrix ‘M’, the character ‘c’ has the indices {2, 5, 7, 18, 23}. Select the index ‘2’ because it is greater than and closer to the index of first character position (see Table 5).

Following the DMA, read the third character of the pattern ‘P’, which is ‘a’, and as we have seen, ‘a’ has the indices {3, 8, 10, 12, 15, 22}, so select ‘3’, which is greater than ‘2’. Finally, repeat the algorithm until reaching the end of the pattern ‘P’, or until an exact match is obtained. Table 6 shows the final result of the first attempt of the array list ‘L’ with its values.

**Table 2.** The index matrix ‘M’ for the text T = ‘ gcatcgcagagagtatacagtacg’.

| Alphabetical characters | Indices of the text characters | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | … | *n* |
| a | 3 | 8 | 10 | 12 | 15 | … | 22 |
| b | . | . | . | . | . | … | . |
| c | 2 | 5 | 7 | 18 | 23 | … | . |
| . | . | . | . | . | . | … | . |
| g | 1 | 6 | 9 | 11 | 13 | … | 24 |
| . | . | . | . | . | . | … | . |
| t | 4 | 14 | 16 | 21 | . | … | . |
| . | . | . | . | . | . | … | . |
| z | . | . | . | . | . | … | . |

**Table 3.** Reading the pattern ‘P = gcagagag’ and creating the array list ‘L’.

| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Array list ‘L’ | g | c | a | g | a | g | a | g |
| First attempt | | | | | | | | |

**Table 4.** Apply step 3.

| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Array list ‘L’ | g | c | a | g | a | g | a | g |
| First attempt | 1 | | | | | | | |

**Table 5.** Apply step 4.

| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Array list ‘L’ | g | c | a | g | a | g | a | g |
| First attempt | 1 | 2 | | | | | | |

**Table 6.** Final result of first attempt of array list ‘L’.

| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Array list ‘L’ | g | c | a | g | a | g | a | g |
| First attempt | 1 | 2 | 3 | 6 | 8 | 9 | 10 | 11 |

**Table 7.** Final result of all matching attempts.

| Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Array list ‘L’ | g | c | a | g | a | g | a | g |
| First attempt | 1 | 2 | 3 | 6 | 8 | 9 | 10 | 11 |
| Second attempt | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

From Table 6, the first attempt shows the positions of the characters for the pattern ‘P’, and also shows that there is a gap between 3 and 6 and between 6 and 8 (positions non-contiguous). The DMA repeats steps 2–5 as mentioned in step 6 to obtain more matching similarities between the pattern ‘P’ and the text ‘T’. The result is shown in Table 7.

As shown by Table 7, the probability of matching increases whenever there are more sequence numbers. Accordingly, the second attempt has the maximum probability of similarity. Meanwhile, the second attempt obtained an exact matching with the pattern ‘P’.

## 4.2. Simulation results

We built a simulation to demonstrate the performance of the DMA and its compatibility with the exact string matching algorithms, and approximate matching algorithms. In addition, this simulation was performed to compare the performance of the DMA with the conventional approximate string matching algorithm (edit distance algorithm). Our simulation ran on a laptop computer with Intel® Core™2 Duo CPU 2900 4M, 4G DDR3 RAM and Windows XP. The results showed high performance of the DMA over the edit distance algorithm.

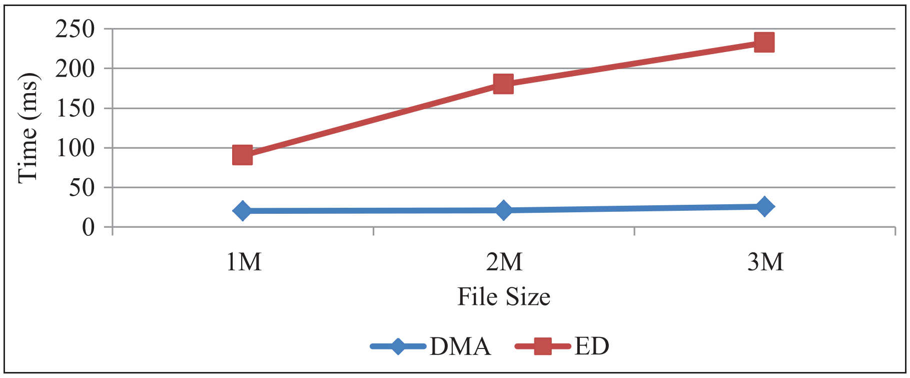

**Figure 1.** Comparison of the execution time between DMA and ED algorithm (best case).

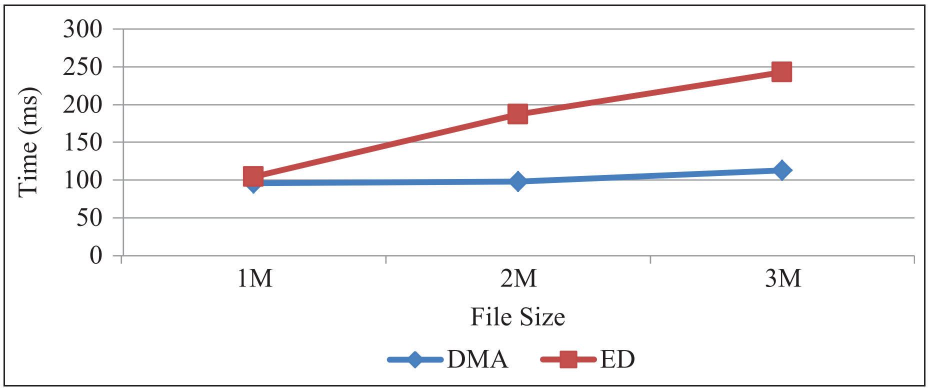

**Figure 2.** Comparison of the execution time between DMA and ED algorithm (worst case).

We carried out three different experiments to search for the pattern p = 'gcagagag' in three text file sizes of 1, 2 and 3 M, respectively. We applied the experiments using the edit distance algorithm, which is one of the best of the approximate strings matching algorithms. The result showed significant improvement in the execution time; applying the DMA decreased the executing time, because the DMA can have direct access to the character position without having to search sequentially.

## 5. Discussion of the DMA performance

In addition, the DMA can also find all the possibile similarities by sorting the results in descending order, where the latest result indicates the maximum percentage of matching or an approximate match between the pattern 'P' and the text 'T'. Also, the DMA is distinguished from the ED algorithm in that the execution time is roughly equal, regardless of the text file size, because of its direct access matching and because it does not depend on the position of the pattern inside the text, at the beginning of the text as best case or at the end of the text as worst case. Figures 1 and 2 depict the improvement in the DMA over the ED algorithm in the best and worst cases, respectively. The time complexity of the DMA is $O(n)$; meanwhile the time complexity of the ED algorithm is $O(n^2)$. Finally, the index matrix 'M' only runs once, as long as no updates are available in the text 'T', meaning that there is no need to create the index matrix 'M' for every 'P'.

Table 8 shows the comparison between the DMA and ED algorithms in terms of the numbers of exact matches and approximate matches for Figures 1 and 2. As shown by Table 8, the performance of the DMA is better than that of the ED algorithm, because in both scenarios, the number of exact matches and approximate matches for the DMA is higher that than that for the ED algorithm, which makes DMA superior to the ED algorithm.

## 6. Conclusion

In this paper, we have proposed a general algorithm, called the DMA, in order to improve the performance of the approximate string matching and exact string matching algorithms. We have applied one of the best approximate string matching algorithms, the Edit Distance algorithm. The results showed good performance in the DMA.

**Table 8.** Comparison between DMA and ED for exact matches and approximate matches.

| | File size | | | | | |
|---|---|---|---|---|---|---|
| | DMA | | | ED | | |
| | 1M | 2M | 3M | 1M | 2M | 3M |
| Number of exact matches | 1 | 1 | 1 | 0 | 0 | 0 |
| Number of approximate matches | 3 | 5 | 9 | 2 | 3 | 5 |

## Funding

We would like to thank Al–Zytoonah University of Jordan - Faculty of Science and Information Technology for its support that enabled us to complete this work.

## References

[1] Abu-Alhaj MM, Abu-Hashem MA, Hnaif AA and Manasrah AM. An innovative platform to improve the performance of exact-string-matching algorithms. *International Journal of Computer Science and Information Security* 2010; 7(1): 225–227.
[2] Navarro G and Fredriksson K. Average complexity of exact and approximate multiple string matching. *Theoretical Computer Science* 2004; 321: 283–290.
[3] Berman KA and Paul JL. *Algorithms: Sequential, Parallel and Distributed*. USA: Thomson, 2005.
[4] Raju SV and Babu AV. Optimal parallel algorithm for string matching on mesh network structure. *International Journal of Applied Mathematical Sciences* 2006; 3(2): 167–175.
[5] Karen R and and Ramsayb J. Now what was that password again? A more flexible way of identifying and authenticating our seniors. *Behaviour and Information Technology* 2007; 26(4): 309–322.
[6] Won J-I, Park S, Yoon J-H and Kim S-W. An efficient approach for sequence matching in large DNA databases. *Journal of Information Science* 2006; 32(1): 88–104.
[7] You J, Park S and Kim I. An efficient frequent melody indexing method to improve the performance of query-by-humming systems. *Journal of Information Science* 2008; 34(6): 777–798.
[8] Raju SV and Babu AV. Parallel algorithms for string matching problem on single and two dimensional reconfigurable pipelined bus systems. *Journal of Computer Science* 2007; 3(9): 754–759.
[9] http://www-igm.univ-mlv.fr/~lecroq/string/