# Improved skip algorithm for single pattern searching:

## Abstract:

We are presenting a novel idea in this paper for a single pattern in strings. Skip search is the idea.

Only half memory locations are available to inspect the pattern for presence in the given text in the

Worst case. A modified version of the characters in the particular string and checking whether these

Characters are the pattern starting and ending characters. This allow to get the overlap of the pattern size and time of search. The algorithm has a low space complexity. The time required for execution is compared to the Naïve algorithm and the Knuth-Morris-Pratt (KMP) algorithm, and the skip algorithm performed better for the majority of the test cases. When the pattern appears at the end of the text or not appear, the algorithm executes even faster. The algorithm is suitable for any pattern matching project.

## Introduction:

The pattern matching algorithm seeks one or more instances of a pattern in the given text. In this article, we suggested a skip algorithm, which is an exact pattern matching algorithm based on Naïve search algorithm with skip positions modified. The algorithm based returns the location of location of the pattern ‘s first occurrence in text.

An examination of the results on exact pattern matching algorithms begins with Knuth-Morris-Pratt algorithm. The majority of the research paper is concerned with reducing the amount of comparisons and processing time. We contrast the skip algorithm’s results to that of the Naïve algorithm and Knuth-Morris-Pratt algorithm. The complexity is compared to a large number of other algorithms listed in the literature survey.

## Related work:

The brute force algorithm or Naïve check contrasts the pattern with any option of moving the text to one end. The Naïve search algorithm has no processing step. The Naïve search algorithm has a time complexity of O(mn), where m is the text size and n is the pattern size.

Knuth _Morris _ Pratt algorithm operates by increasing shift time. This shortens the mothed by making use of prior experience gained from comparisons. This demands pattern preprocessing. This fits best if the pattern is identical.

The comparison from the left to the right pattern Is one of the efficient algorithms published by Boyer_ Moore. The algorithm is known to be one of the most effective algorithms. There are specific rules of moving. The algorithm was pre-processed and in the worst case took extra time. The time complexity of processing is O (m + |∑|) and the worst time is O (nm + |∑|). The algorithm Boyer _Moore_ Hors pool (BMH)raises shift length in the in the text of heuristic occurrence. The time complexity of O (m + |∑|) in this case is the pre-processing and the worst time is O(nm). The fast search(QS) algorithm is another common search algorithm. The scan is kept from left to right and the condition for shift are based on the law of mischanged character. The worst thing is the algorithm. Similarly, the Horspool algorithm is derived from cases complexity. The Boyer -Moore- Smith algorithm(BMS) is a relation to and property that shifts bad character from BMH. The complexity of preprocessing is O (m + |∑|) and the search time is O(mn). Proposed a matching algorithm with calculations and the analysis of a string pattern. Proposed multi-pattern matching skip algorithms and reduced the number of multi-pattern search comparisons. The algorithm of the Boyer-Moore algorithm has been updated and the search speed has increased. Three algorithms have been introduced as general idea. They pre-processed the pattern and developed and used the event list or pattern matching. Suggested a two-way search algorithm. The researchers used the notion of change choice improvement. Combine the correct and the wrong character of the partial text window for the same change duration to the left of the pattern. The algorithm is O(mn)/2 in complexity. However, it requires an additional preprocessing period.
It was suggested that the Naïve search algorithm could be modified and preprocessed more quickly. The algorithm is an advancement in brute strength, which involves both the pattern and the text pre-processing. A contrast is made using Reverse Colossi (RC) as the pre –processed input to the algorithm. The O (m) is a dynamic preprocessing phase, while the quest phase is O(n).

## Algorithm:

The aim is to look more quickly. The suggested skip search also scans the pattern from left to right, similar to more widely used search algorithms such as KMP, BMH, BMS. This algorithm is uniquely characterized by the increasing index by the skipping of undesirable elements.

The text is divided into three categories:

1) The portion of a string that includes elements which may be the beginning of the pattern only.
2) The portion of the string that includes elements which elements which might be the beginning and the end of the pattern.
3) The string element that can only be the end point.

For example;

The string index from 0to3 will only be the starting of a design where the pattern is long 5 (index0to4) and the text is long 40(index0to39). this is part of Group1.The 4to35string index will either be the last pattern character or the initial pattern character. This is Group2.The string from index(36to39) can only be a pattern end character. This is Group3.

In the quest period the access time to the memory plays a major part. The fewer memory places are reached; the less time can be running. If the number of accessed memory locations is lower, the search time will be dramatically decreased. The start character is tested only in Group1 to maximize the search time. The search group2 scans for beginning and end of the design. in the worst case, searches allow one to lower 50% of the accessed memory sites. The last match is tested for the elements of Group alone. Examples are the functioning of the algorithm. May S be the text.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a | a | b | a | c | b | a | b | c | b |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| c | c | a | c | c | a | c | c | b | b |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| c | a | c | b | c | b | c | a | c | a |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| c | c | a | b | c | b | c | a | c | a |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| a | a | b | b | c | c | c | a | c | a |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| b | c | a | b | a | b | c | a | c | b |

And pattern P be

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| a | b | c | a | b |

The string S is split into three
Group 1

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| a | a | b | a |

Group 2

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 4 | 5 | 6 | 7 | 8 | 9 |
| | | | | c | b | a | b | c | b |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| c | c | a | c | c | a | c | c | b | b |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| c | a | c | b | c | b | c | a | c | a |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| c | c | a | b | c | b | c | a | c | a |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| a | a | b | b | c | c | c | a | c | a |

| | | | | | |
|---|---|---|---|---|---|
| 50 | 51 | 52 | 53 | 54 | 55 |
| b | c | a | b | a | b |

Group 3

| | | | |
|---|---|---|---|
| 56 | 57 | 58 | 59 |
| C | a | c | b |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a | a | b | a | c | b | a | b | c | b |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| c | c | a | c | c | a | c | c | b | b |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| c | a | c | b | c | b | c | a | c | a |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| c | c | a | b | c | b | c | a | c | a |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| a | a | b | b | c | c | c | a | c | a |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| b | c | a | b | a | b | c | a | c | b |

For Group1 elements, whether it may be the beginning character, the elements are verified. During quest in the worst case, colored entries are accessed. In the ideal case, you should miss all white entries. All substrates that correspond to the starting character shall be verified with the search algorithm.

```
Algorithm SEARCH
Input:
String S of length l. Stored in an array of index 0 to l-1
Pattern P of length m. Stored in an array of index 0 to m-1
Output:
Print the starting index in the string if the pattern is present (can be between 0 to l-
m).
Return l if the index does not exist
    1.  F=P[0];L=P[m-1];icount=m-1//Store the first and last character of the
        pattern in F and L
    2.  For i=0 to i=m-2// Elements of Group 1
         If(s[i] =F)  SEARCH_FOR _MATCH(i,i+m-1)
    3.  I=i+(m-1)//Initial position of Group 2
    4.  While  i<=l-m //Elements of Group 2
        a.   If(s[i] =F) SEARCH_FOR _MATCH (i,i+m-1)
        b.   If(s[i] =L) SEARCH_FOR _MATCH (i-m+1,i)
        c.   i=i+1;
        d.   icount=icount+1;
        e.   if (icount == m-1)
                  a.   i=i+m-1;//This enables to skip alternate m-1 elements in
                       group 2
                  b.   icount=0;
    5.  For i=l-m+1 to l-1// Elements of Group 3
        If(s[i] =p[m-1]) SEARCH_FOR _MATCH (i-m+1,i)
```

ForGroup2 characters, the elements are checked whether it is the start or end character. The Search for Match algorithm, which is direct and one the comparison of components, is used to further verify all substrates that match either first or last character. Testing only takes place on an alternative (m-1) factor substring. For Group3 elements are examined if it may be the final character

## The Result:

The time necessary to deliver the method is based on the number of accessible memory location.

The KMP method performs more rapidly than this suggested approach if the text with the same sub pattern occurs more than once. Skip method better work if the input characters are positioned randomly.

The Result are compared to the search:
1) Naïve Searching algorithm.
2) Knuth Morris Parrt algorithm.

Naïve search method is used for comparison, as it doesn't require more space or has no pre processing phase.

Knuth–Morris–Pratt algorithms are selected for comparison because they are not as complicated as other algorithms in pretreatment phases, i.e., O(m) and the searcher phase is O(n).

The time required for implementation has been analyzed.

The installation was carried out on a 16 GB RAM Intel i7. The performance was done ten times and the average total results were recorded.
The test cases were determined on the basis of:
1) Text size (The number of characters in the text)
2) Pattern size (the number of characters in the pattern)
3) Alphabet size (the number of different characters used in the text).

The first test data was generated with all possible combinations of:
1) Text size 'L' (characters).
2) Pattern size 'M' (characters).
3) Alphabets 'A' (characters).

**Table 1**

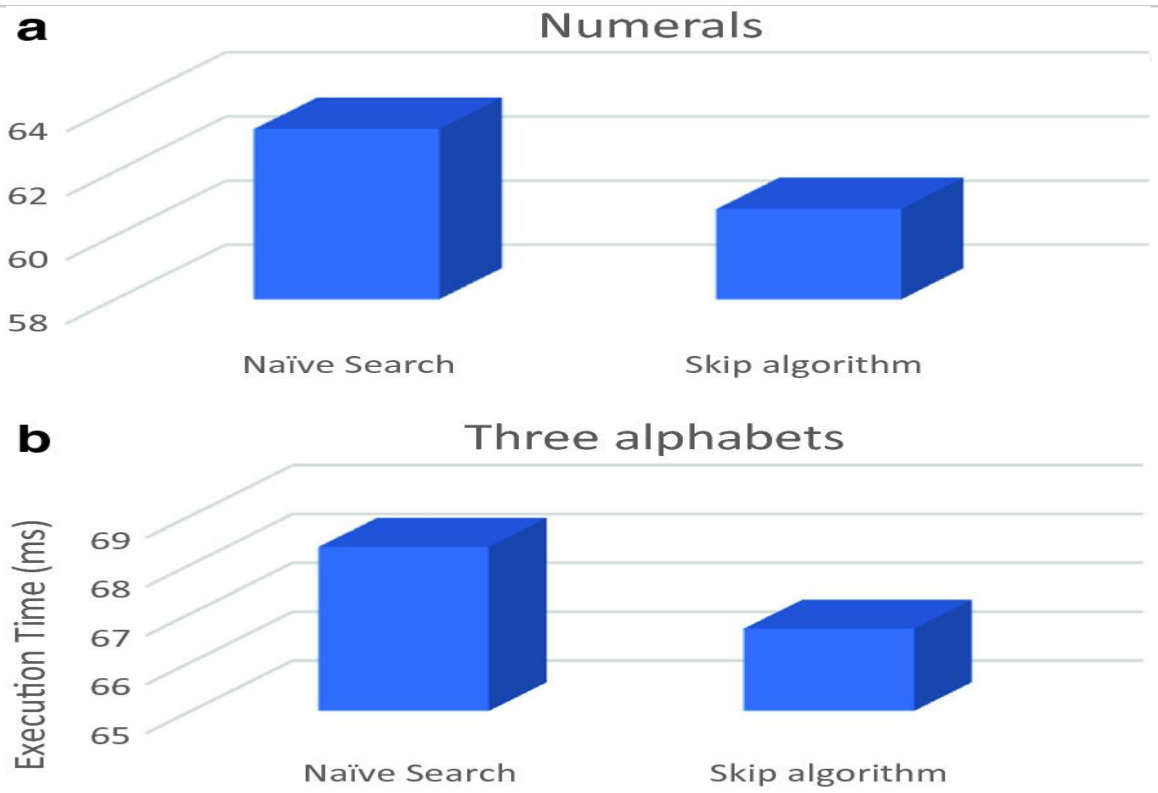
Input parameters for data set 1

| Input | Combinations |
|---|---|
| Text size ‘L’ (characters) | 1 to 15, 16 to 63, 64 to 255, 256 to 2048, >2048 |
| Pattern size ‘M’ (characters) | 1 to 255 depending upon text size |
| Alphabets ‘A’ (characters) | Digits alone (0, 1, 2,..., 9)<br>Only three alphabets 3 (a, b, and c) |

**Table 2**

a Time taken for two alphabet input. b Time taken for numerals s

| Three alphabets | |
|---|---|
| **Algorithm** | **Average time(ms)** |
| (a) | |
| Naive search | 68.36 |
| Skip algorithm | 66.69 |
| (b) | |
| Naive search | 63.32 |
| Skip algorithm | 60.8 |

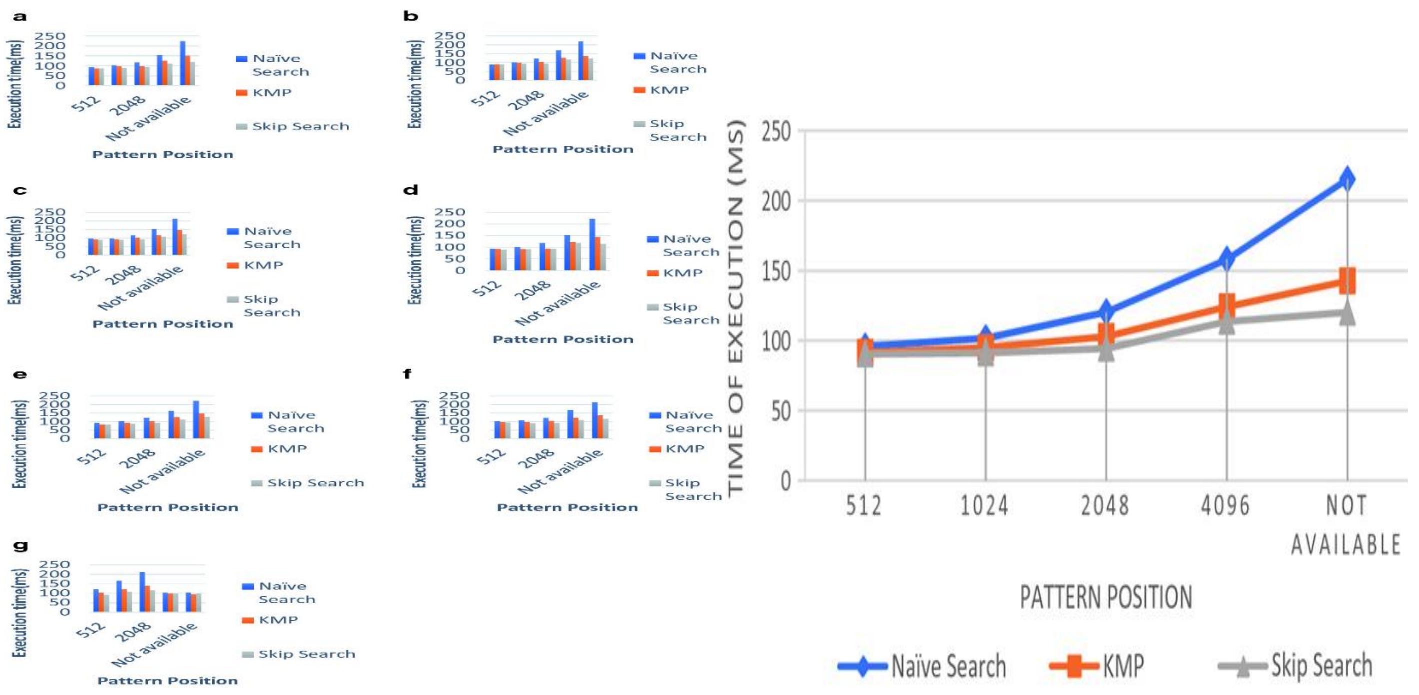


For a fixed input size of 8062 characters, the second data were created for every feasible combination of pattern sizes and placements. The parameters used are shown in Table3.

**Table 3**

Input parameters for data set 2

| Input | |
|---|---|
| Pattern size ‘M’ (characters) | 16, 32,64, 128, 256,512,1024 |
| Position of the pattern in text | 512, 1024, 2048, 4096, not available |



The complexity of times is directly proportional to the amount of items that the provided string accesses. O(mn/2) is the average time complex. The best scenario is when the search pattern is at the beginning O (1). The worst situation is not to detect the pattern. The average complexity in this case is O(mn/2). Only the space for the pattern and string are needed for the algorithm. No further area is needed. It's O(m+n).

**Table 5**

Complexity analysis

| Algorithms | Preprocessing phase | Searching space | Extra space |
|---|---|---|---|
| Naïve search | – | $O(mn)$ | – |
| Knuth–Morris–Pratt (KMP) | $O(m)$ | $O(n)$ | $O(m)$ |
| Boyer–Moore (BM) | $O(m + \|\sum\|)$ | $O(mn)$ | $O(m + \|\sum\|)$ |
| Boyer–Moore–Horspool (BMH) | $O(m + \|\sum\|)$ | $O(mn)$ | $O(\|\sum\|)$ |
| Quick search (QS) | $O(m + \|\sum\|)$ | $O(mn)$ | $O(\|\sum\|)$ |
| Boyer–Moore–Smith (BMS) | $O(m + \|\sum\|)$ | $O(mn)$ | $O(\|\sum\|)$ |
| Reverse Colussi (RC) | $O(m^2)$ | $O(n)$ | $O(m + \|\sum\|)$ |
| Bidirectional | $O(m + \|\sum\|)$ | $O(mn)/2$ | $O(m)$ |
| Skip search | – | $O(mn)/2$ | – |

The quality was 100% in all the test dataset. Test1. The analysis procedure for three alphabet inputs and numbers was checked. The algorithm Skip did best than the algorithm Naïve Search. As random text is engaged in the real use of the technique, test 2 was chosen. The algorithm for skipping has done better than the Naïve and the KMP. Caused by long search position, the time needed to execute rapidly fell. So, if the search pattern is not located in the text provided the ideal result for the search algorithm is to skip. The search of skips aligns faster than the other methods. In some instances, it is possible to skip the m - 1 items before the last m − 1. You can detect this and save more time. For many patterns query tasks, the method may be further refined. Mental graphs are deployed here because they allow people to know quickly what they should do in the case of a calamity. Disaster incidences are connected with certain relationships. Whenever the relationship exists, the linkage connects.