Multi-Threaded Approximate Pattern Matching Based on Edit Distance

Shivani Jain[#1], Dr. A.L.N. Rao[#2], Ankit Jain[#3]

[#1 IT], [#2 IT],[#3IT] MTU, MTU, MTU
Vidya College of Eengineering, Meerut, UP, India
Galgotia Engineering College, Greater Noida, UP India
Vidya College of Eengineering, Meerut, UP, India
[1]*shivanij_1110@yahoo.com*
[2]dr.rao@aol.com
[3]*jain.ankit11@yahoo.com*

Abstract— **This research article concerned with approximate string matching. It also describes the importance of design of efficient "Approximate Pattern Search Algorithms in molecular database". Approximate Search, which allows matching patterns that do not exactly match the search term, but allows some errors between the pattern and the text. This paper uses the Multithreading concept for attaining speed on a multi core system, such that this string matching approach will run parallel. The java technology will be used for using the multithreading concept. Matching is done that is based on edit distance (Levenshtein Algorithm) using the concept of multithreading in a multi core system for providing fast execution. Divide the text into segments then apply the Levenshtein Distance algorithm on each segment concurrently, using the multithreaded concept. This pattern matching algorithm performs the parallel string searching on different text segments by executing a number of threads simultaneously. This approach is advantageous from all other string-pattern matching algorithms in terms of time complexity. Therefore this procedure improves the efficiency of approximate string matching and gives the near-optimal results.**

Keywords— **Multithreading, Edit Distance, Levenshtein Distance, Approximate String Matching**

I. INTRODUCTION

String matching is one of the most important problems in typical string algorithms, with applications to text searching, biological applications, pattern recognition etc. String matching or searching algorithms try to find places where one or several patterns are found within a larger text. When the pattern is in a single string the problem is known as string matching, locate all occurrences of a pattern P of length n in a text T of length m.

The general goal is to perform string matching of a pattern in a text where one or both of them have suffered some kind of undesirable corruption. Some examples are recovering the original signals after their transmission over noisy channels, finding DNA subsequences after possible mutations, and text searching where there are typing or spelling errors [5].

The problem of finding exact or non-exact occurrences of a pattern P in a text T over some alphabet is a central problem of combinatorial pattern matching and has a variety of applications in many areas of computer science [1].

String searching algorithms can be accomplished in two ways:

1. Exact match, meaning that the passages returned will contain an exact match of the key input.

2. Approximate match, meaning that the passage will contain some part of the key word input [9].

Approximate matching is a challenging problem even if only one error is allowed [6]. The simplest solution is to search the suffix tree of S for every 1-error modification of the query pattern, this requires $O(m^2 + occ)$ time [2].

Approximate String matching is a problem to search for strings similar to a given pattern from the input string [10]. Approximate String matching is one of the main problems in classical string algorithms with applications to text searching, biological applications, pattern recognition etc.

The pattern-matching algorithms being used in the current scenarios match the pattern exactly or approximately within the text. An exact pattern-matching is to find all the occurrences of a particular pattern *(P)* $p_1\ p_2 ...\ p_n$ of m-characters in a text *(T)* $t_1\ t_2\ ...\ t_m$ of n-characters which are put up over a finite set of characters of an alphabet set.

The purpose of string searching is to find the location of a specific text pattern within a text body (e.g., in a sentence, in a paragraph, in a text book, etc.). In string matching algorithms, it is required to find the occurrences of a pattern in a text.

Most of the work focused on the edit or Levenshtein distance d, which counts the number of differences between two strings, that is, the number of character insertions, deletions, and substitutions needed to make the strings equal. This distance turns out to be sufficiently powerful to model many relevant applications (e.g., text searching, information retrieval, computational biology, transmission over noisy channels, etc.), and at the same time sufficiently simple to admit efficient solutions (e.g., *O(mn)* and even *O(kn)* time) [8].

Approximate string matching is a recurrent problem in many branches of computer science, the methodology of finding string patterns within a larger text is known as string matching, locate all occurrences of a pattern *P* of length m in a text *T* of length n. Approximate string matching consists in finding all approximate occurrences of pattern *P* in text *T*.

The Approximate string matching problem is to find all of those positions in a given text which are the left endpoints of substrings. The problem of approximate string matching is typically divided into two sub-problems: finding approximate substring matches inside a given string and finding dictionary string that match the pattern approximately. The string matching problem is to find out a pattern in a text (another string). In approximation string matching algorithm substring is matched approximately within the large string.

Fig-1: Locate all occurrence of the pattern *P* in a text *T*.

String Matching Example

"you write always research paper, my research paper, for paper, I am writing a paper"

paper?

"you write always research **paper**, my research **paper**, for **paper**, I am writing a **paper**"

II. LEVENSHTEIN OR EDIT DISTANCE

The Levenshtein distance [7] is a string metric for measuring the amount of difference between two sequences. The term edit distance is often used to refer specially to Levenshtein distance. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character in the simplified definition, all the operation cost is 1. It is named after Vladimir Levenshtein, who considered this distance in 1965. Time complexity of this algorithm is *O(mn)*. It may not be useful while comparing long query strings. In this research, divide the large text sequence into segments then apply the edit distance to each segment for finding the Approximate all occurrence of the pattern in the text sequence.

The Levenshtein distance is also called Edit Distance. The edit distance $\delta(p, t)$ between two strings *P* (pattern) and *T* (text) ($n = |p|$, $m = |t|$) is the minimum number of insertions, deletions and replacements to make *p* equal to *t*.

To define the edit distance between two strings, let $A=a_1 \ldots a_m$ be any string over an alphabet $\sum$ and let the possible editing operations on *A* be [4]:

- **Deletion-** Deleting a symbol from any position, say *i*, to give $a_1 \ldots a_{i-1} a_{i+1} \ldots a_m$.
- **Insertion-** Inserting a symbol $b \ \varepsilon \sum$ at position *i*, to give $a_1 \ldots a_i b a_{i+1} \ldots a_m$.
- **Replacement-** Changing a symbol at position *i* to a new symbol $b \ \varepsilon \sum$ to give $a_1 \ldots a_i b a_{i+1} \ldots a_m$.

Computing D(A, B) becomes considerably simple as soon as we may assume that there is always an editing sequence with cost D(A, B) converting *A* into *B* such that if an element is deleted, inserted or changed, it is not modified again. This means that all editing operations could be applied on *A* in one parallel step yielding *B* [12].

Example

Pattern = sattern

Text = sateen

```
          1  2  3  4  5  6  7

Text:     s  a  t  e  e  q  n

          |  |  |     |     |

Pattern:  s  a  t  t  e  r  n
```

δ(t, p) = 2

The requirement is to compute a matrix D[0..m, 0..n], where $D_{i,j}$ represents the minimum number of operations needed to match $p_{1..i}$ to $t_{1..j}$ [5].

This is computed as follows:

D[i, 0] = i

D[0, j] = j

D[i, j] = min{D[i − 1, j] + 1, D[i, j − 1] + 1, D[i, j] + δ(pi, tj)}

δ(p, t) = D[m, n]

Fig- 2: Computed Edit Distance with error 2

		0	1	2	3	4	5	6	7
			s	a	t	t	e	r	n
0		0	1	2	3	4	5	6	7
1	s	1	0	1	2	3	4	5	6
2	a	2	1	0	1	2	3	4	5
3	t	3	2	1	0	1	2	3	4
4	e	4	3	2	1	1	2	3	4
5	e	5	4	3	2	2	1	2	3
6	n	6	5	4	3	3	2	2	2

II.

III. MULTI-THREADED APPROXIMATE STRING PATTERN MATCHING

A problem with single-threaded applications is that lengthy activities must complete before other activities can begin (actions execute one after another.) In a multithreaded application, Multithreading allows two parts of the same program to run concurrently [3].

The evolutionary development of processor technology along with the other major enhancements done in the processing methodologies have reduced the searching response significantly.

Multithreading allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run its tasks in parallel mode on a single or multi processor system. CPU performs context switching between threads and it seems that threads are executed at the same time. Najib Kofahi and Ahmed Abusalama proposed a multithreading text search approach to improve search performance at a single CPU machine. The idea is to have multiple threads that search the text from different positions. The pattern may occur at any position, having more than one search is better than searching the text sequentially from the first character to the last one. The first thread examines the first character of its assigned text part, CPU makes a context switch to the second thread to check the first character of its part and so on. This process is repeated until the whole text is examined by all the threads.

Multi-threading is a widespread programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. However, perhaps the most interesting application of the technology is when it is applied to a single process to enable parallel execution on a multiprocessing system.

1) Advantages of Multithreading

The advantage of a multithreaded program allows it to operate faster on computer systems that have multiple CPUs:

- Since the process is divided into many light weight threads. The task gets simplified providing much better performance in process handling.

- Along with multi-threading multi-processing will provide a fast and quick simultaneous execution of various processes in the system.

- Multi-threading also provide optimum utilization of all the computing resources of the CPU, thereby increasing the overall execution speed and providing best utilization of the CPU.

- Multi-threading improve the performance on multi-processor machines.

2) A Parallel Working Approach toward Approximate String Matching

In this approach, multithreading concept uses the multi-core-processor. The threads will execute on multi processor simultaneously.

A direct implementation of parallel approximate pattern matching is to divide the text into multiple segments, each segment is processed by a parallel thread. The thread approximate string matching example is shown in **Fig-3**, using a single thread to find the pattern ***"BA"***, which takes 24 cycles. Then using four threads, it takes only six cycles to find the same pattern. Therefore multithread approach is beneficial in terms of performance thereby increasing the efficiency of the matching process and providing much better and enhanced execution service.

Fig-3: Single vs. Multiple Thread Approach

B B B B B B B B B B B B B B B B B B B B B B **B A**

1 thread 24 cycles

A Single Thread Approach

B B B B B B B B B B B B B B B B B B B B B B **B A**

4 thread 6 cycles

Multiple Threads Approach

IV. OVERVIEW OF IMPLEMENTATION DETAIL

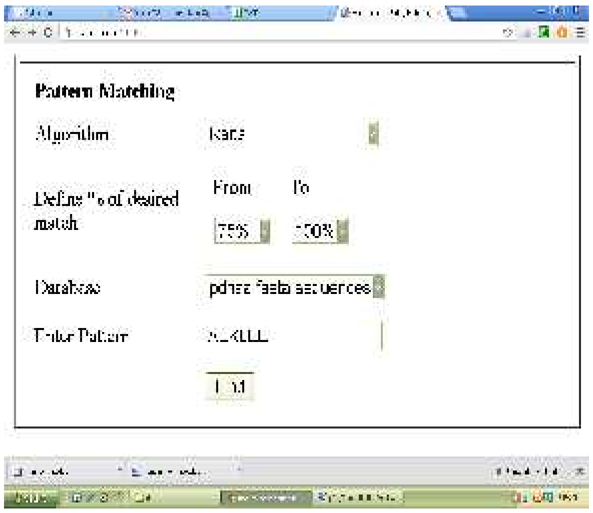
The concept is that a very large size string will be divided in segments depending upon the pattern size. The same pattern will be executed in all the segments of the text in parallel which will largely help to reduce the time complexity of the algorithm. As in terms of memory and processors, this approach is much reliable since multiple executions can be done simultaneously. The same concept is applied here the edit distance for matching the pattern in the text which are divided into multiple segments and are executed concurrently. The major advantage of applying parallel processing is excellent relative time complexity.

For implementing parallel processing, the java technology is being utilized dominantly. Java is unique among popular general-purpose programming languages which makes concurrency primitives available to the applications programmer. The programmer specifies that applications contain threads of execution, each thread designating a portion of a program that may execute concurrently with other threads. Multithreading gives the Java programmer some of its powerful capabilities that are not available in PHP [3].

1) Approximate Pattern Matching Implementation Concept in PHP

First the "Approximating String Matching" approach is implemented in PHP [11] as shown in Fig-4 and Fig-5, but during the course of analysis the implementation came to a halt since PHP does not support Multi-threading so then the transformation was made to the concept implementation in java technology.

Fig-4: Interface of Approximate Pattern matching Algorithm in PHP



Pattern Matching

Algorithm	Raita	
Define % of desired match	From	To
	75%	100%
Database	pdnaa fasta sequences	
Enter Pattern	ALKLLL	
	Find	

The Fig-4 compare the running time between different already existing pattern matching algorithms using PHP as the interface. This was the initial work, therefore used only already existing string matching algorithms with Levenshtein distance.

The user chooses the desired Exact Matching Algorithm like Boyer Moore or Raita then chooses desired percentage matched. After that the database and then enter the pattern.

The pattern now gets matched with the patterns in the database and their approximate percentage is calculated with help of Levenshtein Distance Algorithm. If the string in database gets exactly matched with the pattern entered by the user that is the output percentage is 100% then the output string goes to the algorithm selected by the user initially and after that running time is calculated. With the help of Levenshtein distance calculate the distance means k-errors between the two strings. In Fig-4 match the pattern with the database records one by one with the help of Levenshtein Distance, and then calculate the percentage of matching. If the program calculates 100% matching then it call the user selected Exact String Matching Algorithm.

Using the Approximation method the users get relevant information after going through the several results produced by approximation method, but in case of exact method users have no choice if the patterns are not exactly matched.

Fig-5: Demonstration of the result with best match percentage and running time in PHP

Algorithm	Raita
Desired Percentage	75% to 100%
Database	pdnaa fasta sequences
Pattern	TTETLHG

Result: Highlited pattern using Raita algorithm

And percentage using Levenshtein or Edit distance

Id	**Text**	**Best Match With Percentage**
2	MDKKSARIRRATRARRKLQELGATRLV VHRTPRHIYAQVIAPNGSEVLVAASTVE KAIAEQLKYTGN	85.71%
3	MQAIKCVVVGDGAVGKTCLLISYTTNA FPGEYIPTVFDNYSANVMVDGKPVNLG LWDTAGQEDY	85.70%
5	MADITLISGSTLGGAEYVAEHLAEKLEE AGF**TTETLHG**PLLEDLPASGIWLVISST HGAGDIPDNLSPF	100%

Last timestamp= 027192700.1357007090 Macro Sec

First timestamp= 027175800.1357007090 Macro Sec

Time= 0.00016899999999997 Macro Sec

Fig-5 shows the all patterns that are found according to desired percentage match. The highlighted string from the database matches exactly with the pattern entered by the user and therefore its running time is calculated and shown.

For calculating the running time, procedure has taken the system timestamp as reference before the execution of algorithm and again procedure has taken the system timestamp whenever the algorithm is completed. So the difference between the two timestamp is the desired running time.

The problem can be stated as follows:

Considering a short pattern *P* of length *n*, a long text *T* of length *m*, a maximal number of errors *k*, and all text positions *j* such that a suffix of *T* matches *P* with at most *k* errors (insertions, deletions or replacements).

2) Approximate Pattern Matching implementation Concept in java technology

"Approximating String Matching" approach is implemented using the concept of Java since it requires multiprocessor execution capabilities which can only be incorporated with the help of Multi-threading.

Example

To check the planned algorithm, Fig-6 has been considered for the execution

Fig-6: String *S* and Pattern *P* with length *m* and *n* respectively is taken under consideration.

String S m = 24

ABCDCDDAABBABDCDCBAACCDD

Pattern P n = 6

ABBCBD

Step 1: Considering a String *S* of size *m* and Pattern *P* of size *n* the length is evaluated.

Step 2: Divide the string into number of segments, each segment is equal to the pattern *P* of size *n* and apply Levenshtein distance on each segment simultaneously, for finding closest segments, as shown in Fig-7.

Fig-7: Partition Process (Break the String *S* into number of segments equal to the pattern *P* of size *n*)

1. ABCDCD	2. BCDCDD
3. CDCDDA	4. DCDDAA
5. CDDAAB	6. DDAABB
7. DAABBA	8. AABBAB
9. ABBABD	10. BBABDC
11. BABDCD	12. ABDCDC
13. BDCDCB	14. DCDCBA
15. CDCBAA	16. DCBAAC
17. CBAACC	18. BAACCD
19. AACCDD	20. ACCDDD

As shown in Fig-7, first divide the text into segments and then find the closest segment using the levenshtein distance. Levenshtein algorithm calculate the *k* distance of each segments, utilizing the concept of multithreading on a multi-core system, by utilizing the multi-threads the performance will increase.

Step 3: Print the sequence with highlighted closest segment as demonstrated in Fig-8.

Fig-8: Demonstration of the result after the analysis of the multi-threaded approach

Input

T=ABCDCDDAABBABDCDCBAACCDDD

P=ABBCBD

Output: Highlighted part is the all occurrences of pattern matching approximately.

Search result

ABCDCDDA**ABBABD**CDCBAACCDDD

V. CONCLUSIONS

This paper described the concept of approximate string matching algorithms. String matching or searching algorithms try to find places where one or several patterns are found within a larger text. This approach focused on approximate string matching algorithms based on edit distance. The purpose is implementing an approximate way using Levenshtein distance. With this approach users are able to arrive at certain predictable results. Multithreaded implementation of pattern matching algorithm performs the parallel string searching on different text segments by executing a number of threads simultaneously. This approach is advantageous from all other approximate string-pattern matching algorithms which has been inferred to after the analysis.

REFERENCES

[1] Badoiu M. et al. 2004, "Fast Approximate Pattern Matching with Few Indels via Embeddings," in Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms, Louisiana, pp. 651-652, 2004.

[2] Cobbs, A. 1995, " Fast approximate matching using suffix trees. In: Proceedings of Symposium on Combinatorial Pattern Matching" , pp. 41–54

[3] Deitel P. and Deitel H. 2003, "Java How to Program", Prentice Hall.

[4] E UKKONEN 1985, "Algorithm for Approximate String Matching", information and control, Elsevier

[5] GONZALO NAVARRO 2001, "A Guided Tour to Approximate String Matching", ACM Computing Surveys, Vol. 33, No. 1, March 2001, pp. 31–88.

[6] Ho-Leung Chan et al. 2010, "Compressed Indexes for Approximate String Matching", Springer, Volume 58, Issue 2, pp 263-281

[7] LEVENSHTEIN, V. 1965, "Binary codes capable of correcting spurious insertions and deletions of ones", Probl. Inf. Transmission 1, 8–17.

[8] Marcos Kiwi et al. 2011, "On-line approximate string matching with bounded errors", Theoretical Computer Science 412 (2011) 6359–6370

[9] Mhashi M. et al. 2005, "A Fast Approximate String Searching Algorithm," Computer Journal of Science Publication, vol. 1, no. 3, pp. 405-412.

[10] P.A.V.Hall, G.R.Dowling 1980, "Approximate String Matching", ACM Computing Surveys,12,4, pp.381-402.

[11] Shivani Jain#1, Dr. A.L.N. Rao 2012, "Different Pattern Matching Algorithms with Molecular Sequence in PHP", IJAIR, ISSN: 2278-7844

[12] Wagner and Fisher 1974, "The string to string correction problem", J. Assoc. Comput. Mach. 21, 168-178.