International Journal of Academic Research in Computer Engineering
Print ISSN:2476-7638 and Online ISSN: 2538-2411
Vol. 1, No. 2, Pages. 73-77, November 2016

International Journal of Academic Research in Computer Engineering
Volume 1, Number 2, Pages 73-77, November 2016
www.ijarce.org
IJARCE

Algorithms for Approximate String Matching

Zafar Ali[1*]

1. Department of Computer Engineering, Federal Government University, Virtual University of Pakistan, Pakistan.

Receive Date 2016.08.15; Accepted Date: 2016.11.14, Published Date: 2016.11.15

*Corresponding Author: Z.Ali (zafaralikhan14@gmail.com)

Abstract
String matching plays a major role in our day to day life be it in word processing, signal processing, data communication or bioinformatics. Approximate string matching is a variation of exact string matching that demands more complex algorithms. As the name suggests, in approximate matching, strings are matched on the basis of their non-exact similarities. Quantification of the parameter "similarity" and its efficient computation received a lot of attention in the recent past. Many definitions of similarity have been given depending on application needs. One of the standard ways of quantifying similarity is using the well-known edit distance. It is defined as the number of edits (insertions, deletions, and substitutions) needed to go from one string to another. If similarity or distance has a fuzzy co-efficient where "Hat" and "Fat" are considered the closest words due to the adjacency of the alphabet keys H and F to G. In a similar way, fuzzy concepts can be applied for a DNA sequence similarity where evolution is rule based. Many algorithms based on binary tree or improvement on it as suffix tree, dynamic programming, indexing of strings before searching are being used. Neural network gives a very good result as it is efficient to learn and eliminate case based errors. Further a fusion of fuzzy sets at the input, output and the neurons can greatly improve the matching process.

Keywords: string matching, edit distance, DNA matching, neural networks

1. Introduction

String matching finds wide application in our day-to-day life [1, 2]. The huge number of conferences and journals dedicated for this specialized area shows the need for research for this application oriented area. Initial research started with exact string matching. Subsequently it was found that approximate string matching has more applications and more algorithms intensive. As the name suggests approximate matching means the searched string and the found string are approximately the same. Often small difference occurs due to error. Often error correction is also needed in the algorithm. Strings usually contain symbols from natural language alphabets such as English but in other applications binary strings or DNA strings over the alphabet A, C, G, T are used. To make it compact these binary strings are compressed to string of alphabets. In such cases an efficient compression algorithm is required. As the algorithms are application oriented and often an ensemble of algorithms dynamic programming is very efficient in this area.

It is essential to quantify the similarity of the string pair. For this "distance" metric is used. Distance between two strings is estimated by the minimum number of edits (insertions, deletions, substitutions) needed in one string to get the other string. But this simple definition is not enough when widely different applications are considered. Suppose a word "Gat" is typed in the key-board by mistake. The word-processor suggests correct words with distance 1 as "Hat", "Fat", "Bat" and so on. If distance is calculated using fuzzy parameters "Hat" and "Fat" have the least distance due to the proximity of the keys H and F to G. So the more efficient algorithm suggests the correct word as

"Hat" or "Fat". Further an individual is prone to make a particular type of mistake. Studies may show a particular user has a tendency to press the right key instead of the correct one. Then "Fat" should be with the minimum distance and the suggested correct word. This is because the wrong key G is directly to the right of the key F.
The definition of distance becomes quite different in Optical Handwritten Character Recognition (OCR). A person may write "Oat" bit die to incomplete circle it might be recognized as "Gat". So in this case "Oat" has the minimum distance to "Gat".

However, most of the recently developed algorithms use the simpler edit distance definition. In the edit distance along-with alphabet change a positional shift is sometimes considered. This is explained in the next section with an example. The parameter is called difference and is given by the notation 'k'. k=2 means maximum 2 alphabet mismatches are allowed.

Most of the recent research in this area deals with performance or comparison of various algorithms. In some cases time complexities are estimated and compared. For many algorithms comparisons are done with exact run time obtained after execution of the implementations.

Comparisons are done for specific applications. Most common applications are: English word, DNA word, protein word and speech file.

2. Related Work

Hall and Dowling discussed in detail the various applications of approximate string matching [3]. The mismatch might be due to incorporated errors and error correction should be included in the algorithm. Due to incorrect error detection and correction the searched string may get a wrong string. So detection and correction should be done to obtain the best match. Authors applied graph theory to find the best match between two words. In this process, if the two symbols are not matched the path-length becomes 1 between two nodes. For a match the path-length is zero. The shortest path gives the best match.

Landau and Vishkin did complexity studies for a particular case of approximate string matching [4]. A text of n symbols and a pattern of m symbols are matched for symbols that can differ in position and value by maximum k. All n, m and k are integers. They explained the matching process with an example. Let the text be abcdefghi, the pattern bxdyegh. In this case, n=9, m=7 and k=3.

The matching between the pattern and text can be given in Table 1.

Table 1. Matching between the Pattern and Text

Pattern		b	x	d	y	e		g	h	
Text	a	b	c	d		e	f	g	h	i

The task is to find the pattern within the text. For this, b and b are matched. There is a mismatch between x and c and so on. It can be seen that number of mismatches are at 3 places for the pattern alphabets: x, y, and text alphabet f. Since maximum allowed number of mismatches is k=3 the pattern is found in the text.

This type of pattern matching finds wide applications in molecular biology and speech recognition. To accomplish this task both serial and parallel algorithms are developed. The Serial algorithm is a new algorithm proposed by the authors. The time complexity for the common part which is pattern analysis is O(log(m)) for both the algorithms. In the parallel algorithm m2 processors are used for this part. For the second part of final matching time complexity is O(k) for parallel algorithm with the use of n processors. For serial algorithm it is O(n(k+log(n))). Space complexity is O(n) for both the algorithms. It is observed that there is an improvement in time complexity for the serial algorithm when it is compared with other existing algorithms. There is further improvement in time complexity for the parallel algorithm.

The authors further improved the time complexities by development of more efficient algorithms [5]. The improved time complexities are O(log(m)+k) for the parallel algorithm using n processors and O(nk) for the serial algorithm.

Myers developed an algorithm using dynamic programming for a similar application [6]. The original pattern and the text can be in bit-strings. These bits are compressed to symbols in the algorithm. The author provides timing data for different proposed algorithms.

It was observed that BYN algorithm gives best timing for lower values of k. But timing is best for the presently discussed algorithm for moderate to higher values of k. Another interesting feature of this algorithm is; the timing does not vary appreciably with a variation of k. Timings change very fast with a variation of k for the other algorithms. For high value of pattern or alphabet size timing shows a staircase function for this algorithm.

Wu studied string searching using the tool AGREP [7]. Table 2 shows the run-times for AGREP and other comparable tools. Results show AGREP is the best for large text. For small text AGREP comes second after GRE.

Table 2. Exact matching of simple strings [7]

Text size	agrep	gre	e?grep	egrep
1 Mb	0.09	0.11	0.11	0.79
200 KB	0.028	0.024	0.038	0.218

Table 3. Approximate String Matching with AGREP tool [7].

Pattern	Er = 1	Er = 2	Er = 3
'string matching	0.26	0.55	0.76
'matching'	0.22	0.66	1.14

Table 3 gives the run-times for approximate string matching allowing varying errors. Performances of other tools are observed to be ways behind.

Cohen et al. studied approximate matching using an open-source Java toolkit namely, "Second String" [8]. The toolkit is used to match names with existing records. This type of matching is useful for census. In the toolkit the distance between the word pair to be matched can be defined and the value can be edited A very detailed study is done but the results are presented in a bit cumbersome way with too many algorithms and distance definitions.

McCallum et al. developed a clustering algorithm and applied it for multiple patterns matching [9]. The algorithm is based on K-mean clustering and authors named it as "Canopies". The algorithm is useful for paper search in the reference list and it will cluster multiple matched records. Table 4 shows the comparison of the proposed algorithm with the other existing algorithms.

Table 4. Comparison of the proposed "Canopies" algorithm with the other existing algorithms [9].

Method	F1	Error	Prec-ision	Re-call	Minut-es
Canopies	0.838	0.75%	0.735	0.976	7.65
Complete Expensive	0.835	0.76%	0.737	0.965	134.09
Existing Cora	0.784	1.03%	0.673	0.939	0.03
Author / Year Baseline	0.697	1.60%	0.559	0.926	0.03
Naive Baseline	-	199%	1.00	0.000	-

For comparisons shake various metrics are defined by the authors. According to authors, error rate is percentage of pairs of citations that are correctly in the same cluster. For the application correctness means the pair of citation refers to the same paper. The definition given by authors is actually the percentage correctness. The error rate reported in the Table is appeared to be 100 – percentage correctness. Table shows Canopies has the best or the least error rate.

Error rate does not consider the correct cases of different citation pairs in different clusters. This is taken care in the metric Precision. Precision is the fraction of correct predictions among all pairs of citations predicted to fall in the same cluster. Whereas recall is the fraction of correct predictions among all pairs of citations that truly fall in the same cluster. The proposed algorithm has best recall and medium precision. Metric F1 is the harmonic average of precision and recall. Proposed algorithm has the best F1.

Implementation is done in PERL except for existing Cora algorithm which was done in C. It is not clear why the time for the last algorithm is not given. F1 is also not given for the last algorithm. But that is understood. Since F1 is 2/(1/precision+1/recall) it becomes undefined with recall value 0. But precision value 1 and recall value 0 question the use of such algorithm.

The existing algorithm Cora gives over-all appreciable metrics. The existing algorithm Cora is not a clustering algorithm. It is a word matching algorithm. It is not clear why some of the metrics are given in fractions and others in percentage.

Navarro did a very detailed study of the existing algorithms that includes filtering algorithms [10]. He estimated and compared the time complexities of such existing algorithms. The execution time was compared for different situations. Execution time was compared for existing algorithms applied to speech files. Most of the filtering types of algorithms show a better timing for lower value of m. For most of the non-filtering types of algorithms the timing values remain almost constant.

It was observed that algorithms are most efficient for a range of m values. RUS without filtering is the best for m>50. BPM is the best in the range 10 to 50. BPD is the best for m<10. To continue with BPD for higher values of m, filtering should be applied.

Kumar and Duraipandian studied string matching using Artificial Neural Network (ANN) [11]. The search is done with strings over a DNA alphabet. Present algorithm named as Reverse Factor (RF) algorithm, shows minimum number of character comparisons and hence the best of all. The algorithm named as KR algorithm shows number of character comparisons and hence speed of execution is invariant with pattern length. That might be useful in some applications.

It is not clear why the numbers of character comparisons are not integers. Probably the constant value of KR algorithm is taken as 1 and rest of the

values are normalized with respect to the KR value. But usual practice of normalization is to divide by the maximum so that the values get confined between 0 and 1. If the values are checked from the related table one can observe that, the values for KR algorithms are not exactly constant. The value for m=100 is taken for normalization without any known reason.

Table 5. Number of comparisons for different pattern lengths m [11].

m	BF	KMP	BM	RF	KR
1	1.346	1.116	0.301	0.261	1.000
2	1.348	1.109	0.271	0.154	1.000
3	1.335	1.094	0.231	0.110	1.000
4	1.4\342	1.103	0.209	0.087	1.000

Susik et. al. did an interesting study on multiple pattern matching [12]. The algorithm is based on Average Optimal Shift-Or (AOSO) basic algorithm. On this basic algorithm many other algorithms were reported earlier e.g. Fast AOSO (FAOSO), Average Optimal Shift-Add (AOSA). Present algorithm reported by authors is named according to its strategy as: Multiple AOSO on q-Grams (MAG). This algorithm is coded in C++ and executed. The speed is compared with other existing algorithms' code execution. The search is done for 3 types of words: (i) English (ii) Proteins and (iii) DNA.

Execution speeds are compared for different algorithms. For m>8 the results are also compared with MAG-tuned which is MAG with m=8. But it is not clear from the results why MAG-tuned is different from MAG where m=8. Results show MAG is the best or comparable to another algorithm for low number of patterns.

Pang et al. [13] proposed ontological and lexical methods for pulling data from bio-banks. The developed tool, available as open-source, appears to be useful for many biomedical data integrations. Dumais et al. [14] developed an innovative system for information retrieval. The research should get its due attention as it promises improvements of several areas. This includes increasing efficiencies of search engines. Elarian et al. [15] did extensive research on Arabic handwriting recognition system. The system is based on Hidden Markov Classifier. An extensive study in different situations for error rate was made.

3. Analysis

- Recent research focuses on matching between two ensembles with multiple strings. For such applications clustering is an efficient technique
- Efficiency of a string matching algorithm is not absolute. Every algorithm is efficient for a range of string length and difference k.
- Applications may require other algorithms e.g. string compression, error correction along-with string matching algorithm. In this case, the required algorithm is a combination of many algorithms. Dynamic programming concept is very useful in such a case because its breaks the problem into basic blocks and then solves these problems non-recursively as well as save the results in the Table 5
- Exact string matching is utilized as a part of hunt of any event of a string in string B. These calculations are connected in science, and particularly in the section concerning DNA chains.
- Quite a bit of information handling in bioinformatics includes in somehow perceiving certain examples inside DNA, RNA or protein successions.
- On the off chance that the rest of the example P and rest of the content T are equivalent, at exactly that point we contrast the content and example generally there is no requirement for correlation.
- This is likewise investigated that multi-design string matching emerges in various applications including system interruption location, advanced legal sciences, business examination, and characteristic dialect preparing.
- Proficiency of a string matching calculation is not outright. Each calculation is productive for a scope of string length and distinction k.
- Applications may require different calculations e.g. string pressure, blunder adjustment alongside string matching calculation. For this situation, the required calculation is a mix of numerous calculations.

4. Conclusion

In this paper we considered approximate string matching, providing brief glimpses of some papers with algorithms for this problem. In particular, we discussed approximate string matching gets more research attention as it is algorithm intensive and it finds wider applications. String matching can be for two ensembles with multiple strings. For such situations clustering algorithms are useful. There are some more conceivable regions in which string matching can play a key part to excel. String matching technique combines several algorithms depending upon the application. For this reason, we particularly discussed dynamic programming concept that is very useful for such techniques. Dynamic programming is a powerful technique that can be used to solve many problems to reduce the time. From the many algorithms and different application areas, as well as ever increasing amounts of data we can infer that approximate string matching is likely to stay an active research area.

References

[1] H. Hyyro, Practical Methods for Approximate String Matching, Academic Dissertation, University of Tampere, Finland, December 2003.

[2] G. Navarro, Approximate Text searching, PhD Thesis, University of Chile, December 1998.

[3] P.A.V. Hall and G.R. Dowling, Approximate string matching, ACM Computing Surveys, CSUR. Vol. 12, No. 4, pp. 381- 402, December 1980.

[4] G. M. Landau and U. Vishkin, Introducing efficient parallelism into approximate string matching, 18th ACM Symposium on Theory of Computing, pp. 220-230, 1986.

[5] G. M. Landau and U. Vishkin, Fast parallel and serial approximate string matching, Journal of Algorithms Elsevier, Vol. 10, pp. 157-169, 1989.

[6] G. Myers, A Fast Bit-Vector Algorithm for Approximate String Matching Based On Dynamic Programming, Journal of the ACM. Vol. 46, No. 3, pp. 395- 415, May 1999.

[7] S. Wu and U. Manber, AGREP: A Fast Approximate Pattern Matching Tool, Usenix Winter 1992 Technical Conference-San Francisco, pp. 153 -162, January 20-24, 1992.

[8] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, A Comparison Of String Metrics For Matching Names And Records, KDD Data Cleaning Workshop, pp. 1-6, August 2003.

[9] A. McCallum, K. Nigam, and L. H. Ungar, Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching, Sixth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD 2000, Boston, MA, USA, pp. 169- 178, 2000.

[10] G. Navarro, A Guided Tour to Approximate String Matching, ACM Computing Surveys, CSUR, Vol. 33, No. 1, pp. 31 – 88, March 2001.

[11] S. Kumar and N. Duraipandian, Artificial Neural Network Based String Matching Algorithms For Species Classification: A Preliminary Study Of Experimental Results, International Journal of Computer Applications, Vol. 52, No. 14, pp. 21-29, August 2012.

[12] R. Susik, S. Gabowski, and K. Fredriksson, Multiple Pattern Matching Revisited, Prague Stringology Conference, PSC 2014, J. Holub and J. Zdarek Eds., pp. 59 -70, 2014.

[13] C. Pang, D. Hendriksen, M. Dijkstra, K. Joeri van der Velde, J. Kuiper, H. L. Hillege, and M. A. Swertz, BiobankConnect: Software To Rapidly Connect Data Elements For Pooled Analysis Across Biobanks Using Ontological And Lexical Indexing, J Am Med Inform Assoc, Research And Applications, Vol. 22, pp. 65 – 75,

[14] S. Dumais, E. Cutrell, J.J. Cadiz, G. Jancke, R. Sarin, and D.C. Robbins, Stuff I've Seen: A System for Personal Information Retrieval and Re-Use, SIGIR 2014 Test of Time Award Winning Paper, ACM SIGIR Forum, Vol. 49, No. 2, pp. 28 – 35, December 2015.

[15] Y. Elarian, I. Ahmad, S. Awaida, W.G. Al-Khatib, and A. Zidouri, An Arabic Handwriting Synthesis System, Pattern Recognition, vol. 48, pp. 849 – 861.