ANSL Algorithm for String Similarity Matching

Nawaphorn Chartbunchachai, Autcha Mutchalintungkul, and Peraphon Sophatsathit
Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Faculty of Science, Chulalongkorn University
Bangkok, 10330, Thailand
Bonear@hotmail.com, jibjoice25@hotmail.com, Peraphon.S@chula.ac.th

Abstract

Information has played a predominant role and intertwined with our daily digital age lives. Transmission at the speed of light is utmost important in information access and retrieval so as to satisfy consumer's needs timely and effectively. However, one of the obstacles in the speed of searching for the needed information is matching keywords or phrases with what is available in the archive, which is by and large a time-consuming process. This research aims to devise an algorithmic procedure employing string similarity to find similar matching in place of exact matching. The proposed algorithm can be further applied to future machine learning research. The results so obtained are conducive toward enhancing other matching algorithms that might be suitable for specific needs and applications.

Keywords: string matching, string similarity matching, string search, neural networks.

1. Research motivation

In all manufacturing process, quality control and product testing before delivery are mandatory due to the inherent erroneous nature of each process. Administration of such process control and inspection will lead to error discovery and rectification. Programming is no different. It requires systematic testing procedures to locate errors as programs become increasingly complex. Fixing those errors will insure correctness to meet the user's requirements. Consequently, the test results will also serve as program quality assurance.

Testing can be carried out in several ways, for instance, manual checking by meticulously reading every line of code, running the program one module at a time to locate bugs, performing static analysis on program source to unveil logical errors, type checking, whitebox and blackbox testing, etc. The presumption of testing requires that test cases be established for complete program coverage. In so doing, complicate test cases can be further dichotomized to simpler test cases since unrelated or redundant cases will be methodically eliminated. Additional rearrangement of test cases will entail thorough program coverage. Details on how to carry out this test process are elucidated in [1].

One compelling question on when enough testing is enough remains unanswered for years. Exhaustive testing is obviously impractical. The culprit, nonetheless, can be attributed to program complexity. Program complexity is an imperative indicator that suggests the extent to which how thorough testing should be performed and when to stop. Measuring program complexity usually involves three aspects, namely, 1) linguistic metrics which concern primarily on program instructions, 2) structural metrics which focus on the relationships among program objects, and 3) hybrid metrics which combine both linguistic metrics and structural metrics [1]. One popular approach is McCabe's metric, a structural metric which derives program complexity from flow graph according to the relation $M = L - N + 2P$. The proposed approach will utilize this metric to generate all relevant test cases that must exceed the resulting complexity. The reason being is that insufficient test cases will yield inaccurate result as all executable or reachable paths may not be fully covered.

Bearing the above rationale in mind, classical exact string matching algorithms have been shifted toward similar matching scheme due to a number of shortcomings for exact match, for instance, known pattern may not be readily available, searching for exact match is time-consuming, and the exact desired pattern may not exist after all. Thus, similar matching has become one of the research mainstays in the area. Unfortunately, the inherent complexity is an NP problem that renders thorough testing impractical. As a consequence, we propose a concise yet unconventional similar string matching algorithm which excludes most elaborate computation schemes but hopefully is efficient to operate on a variety of string matching problems. The ultimate objective is to adapt it in machine learning [4] technology.

2. Related work

Senvar and Bener [9] translate meanings into weights which eventually are converted to the deviation from the vocabulary tree. This technique has been employed to keyword sequencing. Hofmann [7] applies Probabilistic Latent Semantic Analysis (PLSA) in similarity search analysis. Ma, Zhang, and He [8] suggest the use of word grouping to facilitate closest similarity comparison. Balinski and Danilowicz [6] introduce a distance ordering approach based on how far the search document deviates from the ideal one. Regardless of the forms of weight, distance, or meaning conversion, such representations lend themselves to word/document related researches such as partitioning, hierarchical/flat, non-hierarchical clustering, k-means clustering, Buckshot, Fractionation, all of which utilize single word, sentence, and snippet to devise efficient similarity search algorithms that yield the closest result to user's search objective.

A number of existing techniques rest on the aforementioned methodologies such as suffix tree, tries, in particular, finite state automaton based search algorithms [10, 12] and dynamic programming [11], all of which are computationally complex. The proposed algorithm, however, investigates on different aspects such as string relationships, their occurrences, and inherent roles they play in matching considerations.

3. Proposed ANSL approach

The proposed ANSL algorithm is developed to find string similarity patterns in search texts or target strings to replace similar matching [5]. The user specifies either a single or more word patterns. For a short one word string, the algorithm will start comparing one character at a time and determine the similarity based on user specified threshold. In a long string containing more than one word, the algorithm will proceed to compare one word at a time in the order they appear. Each word's pattern similarity is accumulated, whereby the overall similarity value will reach the specified threshold. The algorithm measures the ratio which is determined from character position, the number of matched or similar characters, character transposition, distance between matched characters, and the different between number of characters in source and target strings. The criteria for pattern similarity are as follows:

1. The allowable string patterns consist only of A-Z and 0-9. Special characters are not permitted.
2. Pattern length limits to 20 characters.
3. Similarity value is computed directly from the position of matched characters
4. Characters are grouped and formed not more than 4 words, each of which must abide by the above pattern regulations.
5. In case of multiple matching, only one word in the same sentence is considered.
6. Users must specify similarity percentage (threshold) before comparison commences.

4. Preliminary experiments on similar matching

Since there are no dedicated similarity matching algorithms for specific application and usage, this research has established an empirical formula to determine the percentage of string similarity between two words as follows:

Similarity (in percentage)
*= (no. of matched characters / no. of characters in pattern) * 100*

In practice, it was found that the above formula could not achieve the desired accuracy threshold since the length of target string might exceed source string that would fail to yield approximately 100% similarity result. Moreover, each matched character might not be conventionally ordered from left to right. In order to arrive at an effective word similarity matching, the proposed algorithm incorporates a number of relevant factors into consideration. They are:

Factor 1: Number of matched characters in wrong position
Factor 2: Distance between two consecutive matched characters
Factor 3: Number of unmatched characters in target string

The above three factors must be carefully weighed in accordance with their application domains. Denote w1, w2, and w3 as the weight of factor 1, 2, and 3, respectively. The above formula becomes

Similarity (in percentage)
*= (no. of matched characters / no. of characters in pattern * 100)*
*- no. of matched characters in wrong position * w1*
*- distance between two consecutive matched characters * w2*
*- no. of unmatched characters in target string * w3*

where w1, w2, and w3 can be obtained from a preliminary similarity analysis which is depicted in Table 1.

Note that every target string contains three matched characters, but yields different weight factors ranging from 1-5 based on our empirical test runs. Upon some preliminary trial and error, we arrived at the appropriate weight values to be w1 = 4, w2 = 2, and w3 = 2 as shown in Table 2.

Table 1 weight analysis of string similarity

target string	matched characters in wrong position	distance between two consecutive matched characters	unmatched character in target string
ABC	-	-	-
BAC	2	-	-
CAB	3	-	-
ABDC	-	1	1
ABDEC	-	2	2
ABCD	-	-	1
ABCDE	-	-	2

Table 2 string similarity in comparison to “the” using w1 = 4, w2 = 2, and w3 = 2

target string	matched characte rs in wrong position	distance between two consecutive matched characters	unmatche d characters in target string	similari ty value (%)
the	-	-	-	100
they	-	-	1	98
hate	1	1	1	92
another	-	-	4	92
teach	1	2	2	88
heat	2	1	1	88
technic	1	1	4	86

The above preliminaries are insufficient to accommodate long string or sentence (containing more than one word) similar matching as they yield only percentage of word similarity. The formula accounts for individual word similarity that does not scale up to sentence context. As such, some adjusting factors must be enhanced to fine tune the yield of the modified formula.

Overall similarity (in percentage) =
*(no. of similar words / no. of subpattern * average similarity of all subpatterns)*
*- no. of words having matched characters in wrong position * W1*
*- distance between two consecutive matched words * W2*

where W1 and W2 are weights corresponding to word similarity in target sentence.

Notice that the size of target string is not included in the above computation as in the previous scenario since it can be arbitrarily long. In particular, those strings that might contain many unmatched patterns will result in low overall similarity percentage. As a consequence, string size factor is dropped to avoid any sporadic outcome. From our preliminary findings, the proper weight so obtained are W1 = 4 and W2 = 2. These values could be more reliably approximated by means of neural network technique.

5. Experiments

The experimental design of similarity measurement systematically dichotomizes all possible factorial combinations into a number of steps as follows:

Step 1:apply only factor 1
Step 2:apply only factor 2
Step 3:apply only factor 3
Step 4:apply only factor 1 and 2
Step 5:apply only factor 1 and 3
Step 6:apply only factor 2 and 3
Step 7:apply all factors

Table 3 demonstrates sample computations of similarity weight from target string stored in files. Table 4 summarizes the above experimental steps for all three relevant factors.

Table 3 sample computations of similarity weight

pattern	target string	relevant factor	similarity value (%)
student	students	3	98
were	where	2,3	97
rat	art	1	96

Table 4 summary of similarity results

trial no.	average similarity value		
	file A	file B	file C
1	81	68	76
2	79	65	74
3	76	57	69
4	79	64	73
5	76	56	69
6	75	56	71
7	78	55	73

where test file A, B, and C contain different types of string to be tested, i.e., file A holds long sentences (strings), file B is a C program, and file C holds short strings. Note that the similarity values reduce as the number of factors involved increase, as well as the characteristics of the strings themselves. The inherent characteristics of the strings play an important role in similarity evaluation. For example, strings may contain repeated or similar words that do not introduce any distinction as in the case of file B. Thus, taking only relevant factors into consideration by the proposed ANSL algorithm significantly arrive at more accurate similarity estimation.

Some benefits from similar matching approach are worth mentioning. First and foremost, searching can be performed efficiently faster than exact matching, whereby considerable time saving is attained. As keyword matching no longer gives the full satisfaction results, full-text search has gained its applicability for real life use. The needs for fast, efficient, and effective search schemes and algorithms are inevitably urgent. Thus, the proposed

ANSL algorithm can serve as the first step toward such ultimatum.

6. Summary and future work

The proposed ANSL algorithm furnishes a string similarity matching technique by applying different weights to represent various emphases on pertinent factors. In so doing, no single factor can dominate the search process as users establish their own weights and threshold values. Nevertheless, fixation of pattern designated by the users renders the algorithm less flexible than it should. Future course of action will incorporate the ANSL algorithm to accommodate similarity match given by regular expression. The use of weight factor in computations will be an imperative basis to cover such extension by means of neural networks through Self-Organizing Map [3], Radial Basis Function Networks, Fuzzy Subtractive Clustering, or Rival Penalized Competitive Learning [2], whereby helping derive suitable weight factors. The algorithm will have to be run on standard benchmark data to validate its conciseness on similar matching objective. This in turn can be further applied in medical, literature, and language applications. The eventuality of this research endeavor will lend itself to machine learning that eases the effort burden on the user's part.

7. References

[1] Boris Beizer, Software Testing Tecniques, Second Edition, Van Nostrand Reinhold, New York, 1990.

[2] L. T. Kan, "Rival Penalized Competitive Learning For Content-Based Indexing", A dissertation for the degree of masters of philosophy, The Chinese University of Hong Hong, June 1998.

[3] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin, 1995.

[4] E. Ostertag, J. Hendler, R. P. Diaz, and C. Braun, "Computing similarity in a reuse library system: An AI-Base Approach", ACM Transactions on Software Engineering and Methodology, pp. 205-228, 1992.

[5] W. Pedrycz, "Computational Intelligence as an Emerging Paradigm of Software Engineering", in Proceedings of the Fourteenth International Conference on Software Engineering and Knowledge Engineering: SEKE'02, ACM Press, pp.7-14, 2002.

[6] J. Balinski and C. Danilowicz, "Re-ranking Method based on Inter-document Distances", Journal of the Information Processing and Management, Vol. 41, Issue 4, 2005.

[7] T. Hofmann, "Probabilistic Latent Semantic Analysis", Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval, Berkeley, California, ACM Press, August 1999, pp. 50-57.

[8] Jiangang Ma, Ynachum Zhang, and Jing He, "Efficiently Finding Web Services Using a Clustering Semantic Approach", Proceedings of the 2008 international workshop on Context enabled source and service selection, integration and adaptation: organized with the 17th International World Wide Web Conference (WWW 2008), Beijing, China, April 22, 2008.

[9] Mehmet Senvar and Ayse Bener, "Matchmaking of Semantic Web Services Using Semantic-Distance Information", LNCS 4243, 2006, pp. 177-186.

[10] Ricardo A. Baeza-Yates and Gonzalo Navarro, "A Faster Algorithm for Approximate String Matching", Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching table of contents, Lecture Notes In Computer Science; Vol. 1075, Springer-Verlag, 1996, pp. 1-23.

[11] Gene Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming", Journal of the ACM (JACM), Vol. 46, Issue 3 (May 1999), pp. 395-415.

[12] Heikki Hyyrö and GonzaloNavarro, "Faster Bit-Parallel Approximate String Matching", LNCS, Vol. 2373, Springer-Verlag, 2002, pp. 203-224.