World Applied Sciences Journal 24 (9): 1192-1198, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.24.09.1150

Longest Common Subsequence Based Algorithm for Measuring Similarity Between Time Series: A New Approach

[1,2]Rahim Khan, [1]Mushtaq Ahmad and [2]Muhammad Zakarya
[1]Faculty of Computer Science and Engineering, GIK Institute of Engineering Sciences and Technology, Pakistan
[2]Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan

Submitted: Jul 17, 2013; **Accepted:** Aug 31, 2013; **Published:** Sep 11, 2013

Abstract: In this paper, a robust algorithm is presented for finding similarity between two time series. The longest common subsequence (LCSS) is calculated by avoiding unnecessary comparisons. A time control parameter is introduced to prevent it from matching the 1st element of a time series with the last element of 2nd time series. Its time and space complexity is less than that of dynamic programming based algorithms because it stores only those elements of time series that are part of LCSS. A special feature of LCSS called special longest common subsequence (SLCSS) is also presented. Both algorithms were evaluated and checked over real time data sets and show excellent results for shorter as well as longer time series.

Key words: Time series · LCSS · SLCSS · DPA

INTRODUCTION

Due to the technological advances particularly in sensors, mobile computing and GPS, the volume of data is increasing at an unpredictable manner in many commercial organizations. Nowadays, electronic devices (such as sensors) are doing the job of computer operators feeding their data to a central location simultaneously. For dealing such huge amount of data, new models were proposed in literature but most of them are application specific [1].

Time series data are sequences of observations produced in non random order and measured simultaneously. Wireless sensor networks, ECG signals, scientific as well as engineering experiments, analysis of stock market exchange data analysis, budget analysis, economic growth and weather forecasting etc are its different applications [2]. The sampling rate of time series generation devices is highly dependent on the application [3]. In environmental monitoring application, sensor nodes take readings after an hour or even more. But deploying the same sensor nodes for scientific application, it takes readings after every second or even less and results in a huge amount of data. Analyzing this data is really a challenge for researchers because traditional methods are not applicable to it. Therefore, new methodologies were proposed for mining and querying of time series data. In order to deal such huge amount of data, new techniques for classification, indexing, segmentation, prediction, clustering and anomaly detection were proposed [4].

In time series data analysis, similarity computation is an important issue. Two Sequences S_1 and S_2 are similar if the number of matched symbols is greater than that of non matched symbols. A sequence S_3 is called a subsequence of sequence S_1 if all symbols of S_3 are also symbols of sequence S_1 [5]. Measuring distance between two objects is straight forward and is calculated by either Euclidean or other distance measures [6] However, measuring distance between time series is a complex problem and different methods were proposed for finding their similarity such as indexing [6,7], longest common subsequence [5, 8, 9], all common subsequence [10], dynamic time warping [4] etc. However, every method has its drawbacks such scaling, longer time series, application specific etc. Therefore, a robust and efficient similarity measuring algorithm is always needed that is general and work efficiently on time series of different length. Gap free longest common subsequence calculation, special longest common subsequence (SLCSS), is also required in different application such stock market analysis, agriculture etc. In response to these problems, this paper

Corresponding Author: Rahim Khan, Faculty of Computer Science and Engineering, GIK Institute of Engineering Sciences and Technology, Pakistan.

presents a robust algorithm for the calculation of LCSS between time series. It overcomes different problems associated with other algorithms such as reduction in time and space complexity, increased efficiency for longer time series, scalability issue etc. It is tested over real time data obtained from wireless sensor networks deployed in oranges orchard of our institute, UCI (UCI KDD Archive, 2012) and onset Computer Corporation live data feeds (HOBO U30 Remote Monitoring Systems 2011-12).

A new feature of LCSS called special longest common subsequence (SLCSS) is also introduced. It is an important measure for the detection of various events in different applications. In agriculture, it can be used to determine whether a disease has occurred or not and in stock market data it is used for finding the increase or decrease in stocks during consecutive time intervals. An algorithm is presented for the calculation of SLCSS. The rest of the paper is organized as follow. In section 2 a brief literature review is presented, in section 3 the proposed technique is discussed in detail, in section 4 and 5 algorithms for calculating longest common subsequence and special longest common subsequence are presented, in section 6 simulation results are presented and finally concluding remarks are given.

Related Work: Similarity between two time series is a well known problem and different solutions were proposed in literature. Initially, traditional methods such as Euclidean distance measures were used for estimating similarity between two time series. However, due to its highly sensitive behaviors to outliers, a need for robust distance measures is always a needed. Non-metric based distance measure called longest common subsequence (LCSS) was utilized because of its non sensitivity to outliers. Its primary objective is to evaluate the similarity between two time series. If the ratio of similarity is greater than their dissimilarity then they are considered as similar. The ratio of similarity varies from application to application. Consider two time series S1 and S2 of length m and n respectively. If L is the length of the LCSS between them then their similarity criteria will be

$$\frac{2L}{length(S_1)+length(S_2)} \geq \varepsilon \quad (1)$$

where, ε is the specified threshold value. ε should be less than or equal to the value obtained by multiplying the LCSS length by 2 and dividing it by the sum of time series lengths. ε is allowable dissimilarity between two time series.

Two variants of LCSS are used in literature that is r-variant and LCSS-variant [2]. R-variant is simple than LCSS-variant because its association with LCSS length only but the LCSS-variant is complex.

The dynamic programming based algorithm (DPA) was presented for finding similarity between two strings in [11]. Each and every character of both strings is matched. Let S1 and S2 be the two strings of length m and n respectively. DPA makes m*n comparisons for finding LCSS but its major drawback is excessive work and unnecessary comparisons.

In response to the problem, [12, 13, 14] presented the idea of match-list data structure. It is a case supporting data structure containing information about the positions of matched symbols. Hirschberg *et al*. [12] proposed that the value of LCSS is incremented when string characters at the corresponding location are similar and assign matched value to class k. Excessive work is avoided by utilizing match-lists but the match-list construction cost is linear to n. Its efficiency is reduced when the matched indexed terms are up to a greater extent and LCSS value is large [2].

The idea of dominant matches was presented in [8]. A match is called dominant if there is no other match belonging to same class. The dominant matches are separated from the pool of available matches. It argued that each and every match is not part of LCSS. For larger alphabet size it demolish quickly.

In addition to minimal witness some dominant matches are also discarded if the length of LCSS is known in advance [9]. The matches considered for LCSS are called dominant matches while others are minimal witnesses. Consider two strings S=ABACACD and T= BACATCSD where LCSS length is 4. Once the desire LCSS is calculated, BACA, the remaining matches for c and d is not necessary [9].

A sliding window based solution was proposed by [5, 15] for the calculation of LCSS. It states that two time series are similar if matching and non-overlapping subsequences exist there. It uses L_∞ norm as a distance measure and allows certain degree of non similar subsequences. The LCSS is computed in three steps namely atomic subsequence matching, long subsequence matching and sequence matching. In 1st step all similar gap free subsequence of length ω is identified where ω is the length of sliding window and is a unit of matching. In 2nd step, all similar windows are merged to form the longest common subsequence. In 3rd step, a subsequence is added to the longest match path only if it does not overlap any other subsequence. But the length of window ω and complexity are the problems associated with it.

Wang [10] proposed that complete common information between time series is not held by LCSS only but 2nd, 3rd LCSS and so on also contain some portion of it. Therefore, it is more suitable to consider all common subsequences for finding similarity between time series and is based on dynamic programming concept. Its complexity is high and not very suitable for larger time series.

A greedy approach is a valuable solution for finding LCSS. Initially, matched symbols list is identified in pre processing step. The symbols are selected in a greedy way from the list by providing equal opportunity to every symbol. Let p_i and $_{pi+1}$ represent the position of i^{th} and $i+1^{th}$ symbols in match-list respectively. They are compared and the one which is smaller is added to L. L stores information about the position of symbols that are part of LCSS.

The Nakatsu, Kambayashi and Yajima (NKY) algorithm [16] is thoroughly analyzed and extended to a general purpose NKY-MODIF algorithm in [2]. The original algorithm is refined in different ways. It reduces unnecessary scanning of the input sequences and stores the intermediate results locally. It utilizes lower and upper bond knowledge of LCSS.

Proposed Technique: We propose an efficient algorithm for computing similarity between time series. Two time series are similar if they have enough time ordered subsequences that are similar. The proposed algorithm computes similarity by finding LCSS between them. Consider the time series S and T given in Fig. 1 and their LCSS is ABACDADB. The DPA makes 100 comparisons for calculating their LCSS while the proposed algorithm make a total of 50 comparisons and avoids nearly half of the unnecessary comparisons. Fig. 1 explains the proposed idea for the calculation of LCSS.

Calculation of Longest Common Subsequence: Consider the example given in Fig. 1. The proposed algorithm takes first symbol of time series S and compares it with symbols of time series T starting from the first symbol. It stops when a match occurs i.e. at the 2nd symbol. The proposed algorithm adds the matched symbol to class k, for storing matched symbols and notes its location in T. In second step, it compares 2nd symbol of S with symbols of T starting after previous matched location i.e. from 3rd symbol but no match found for this symbol in T. Then 3rd symbol are compared in the same way until symbols are matched in both time series i.e. at 3rd location. But, before adding it to class k, the time control parameter δ is checked. It is used to control that how far a symbol is

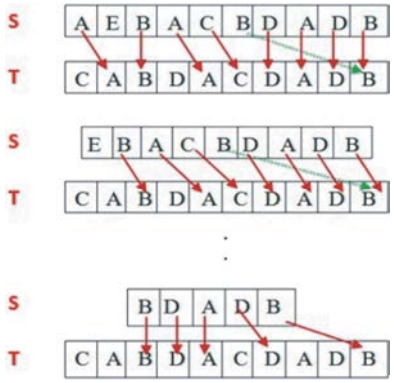

Fig. 1: Proposed algorithm calculation mechanism

allowed to be compared with symbols of 2nd time series. δ is number of symbols from the last matched symbol added to class k to the current matched symbol and it is an application dependent parameter. It is small for applications where the time series are much more similar where large for application containing variations/ dissimilarities in data. If the matched symbols δ value is inside the allowable threshold value then it is added to class k otherwise discarded. In order to understanding this scenario, consider time series given in Fig. 1 where δ value is less than or equal to three. The symbols are added to class k, if its distance from the last matched symbol is less than or equal to three. In time series S, the symbol at 6th location is matched with the last symbol of time series T i.e. B. The last symbol added to class k is 'C' located at 6th location in T. Now B is added to class k only if it satisfies the threshold value test. But it did not qualify the threshold value test because distance between C and B is greater 3. Therefore, B is not added to class k.

The δ value is highly application dependent. In Fig. 1, if the δ value is changed from 3 to 5 then the symbol showed by dotted arrow also qualify the threshold value test and is added to class k. B is the last symbol in T therefore the algorithm stopped there. Similarly, the δ value should not be very small as well. The same procedure is applied for the rest of the symbols.

In second round, the first symbol of time series S is removed and now S contains 9 symbols starting from E as shown in Fig. 1. The algorithm applies the same procedure for calculating LCSS as explained above. The length of LCSS contains in class k is compared with the length of previously stored LCSS. If the former LCSS length is greater than the later LCSS length then stored LCSS is replaced by the former LCSS otherwise discarded. This procedure continues until half of the first time series and stops there because calculation of further LCSS's are not necessary as the length of calculated LCSS is greater

than half of the time series length. However, if the length of LCSS is less than half length of smaller time series then it calculates for more values as well [21].

Calculation of Special Longest Common Subsequence: Longest common subsequence [17] is defined as the common subsequence of maximum length between two time series. LCSS should follow certain rules such as their symbols appeared in one order that is from left to right. But it is not necessary for LCSS's symbols to be appeared at consecutive locations. However, there are certain situations in which it is necessary for the symbols of LCSS to be appeared at consecutive locations in both time series. In agriculture application, crops are affected by various diseases. Most of these diseases occur due to abrupt changes in various parameters such as temperature, humidity, soil moisture, pressure, wind direction etc.

In order to understand this scenario, consider single parameter for the detection of crops diseases that is temperature. Sensor nodes capable of sensing temperature after defined intervals are deployed in agriculture fields. These nodes send their sensed data to a central server called gateway in sensor networks scenario. Gateway is directly connected to a computer on which LCSS based decision support system is implemented. Consider a disease occurs due to continuous changes in temperature that is 22,23,24,25,25 at time intervals t_1, t_2, t_3 t_4, t_5 respectively and the Sensor nodes readings are 22,26, 23,24,23,25,24,23,25, at time t_1, t_2, t_3 t_4, t_5, t_6, t_7, t_8, and t_9 respectively. By analyzing this data, the LCSS based decision support system detects a disease case because it has calculated 22,23,24,25,25 at time intervals t_1, t_3, t_4 t_6, t_9 respectively. But no occurrence found because the conditions are not met. This wrong detection is due to the gaps between symbols of LCSS. Actually, the symbols are matched but they are not at consecutive locations.

For solving this problem, we propose an extended version of LCSS called special longest common subsequence (SLCSS). SLCSS is defined as the longest common subsequence of maximum length between two time series where all of its elements appear at consecutive locations. The LCSS of time series given in Fig. 1 is ABACDADB where its SLCSS is DADB. There are different application areas of SLCSS such as disease detection in agriculture, stock market, habitat monitoring, engineering experiments and weather forecasting etc.

The SLCSS calculation mechanism is the same as that of LCSS but in former we have to take care of additional variables for dealing with consecutive locations. When a match is found we store the location information for both pointers in separate variables [18].

Proposed Algorithm for LCSS: Consider two time series $S = s_1, s_2, \ldots, s_m$ and $T = t_1, t_2, \ldots, t_n$ where m and n are their lengths respectively. TP stores the position of matched elements in time series T. Class K is used for storing the matched elements. Count stores the length of currently maximum LCSS and δ time control parameter. $Count_1$ stores the length of previously calculated LCSS. MD is the smaller time series length divided by 2 and L is the length of shorter time series.

1. while ($L \leq MD$ and $Count_1$<length of m)
2. for i:L to m
3. for j:TP to n
4. if S or T is empty
5. LCSS is zero
6. stop
7. else if(S[i] = T[j] and (TP=0)) then
8. add element to class k
9. set TP equal to j plus 1
10. increment Count by 1 and exit j loop
11. else if(S[i] = T[j] and ($\delta \geq$(j-TP+1))) then
12. add element to class k
13. set TP equal to j plus 1
14. increment Count by 1 and exit j loop
15. else if (S[i] $\neq$ T[j])
16. if(i=m and j=n and LCSS is zero)
17. go out of while loop
18. else
19. move pointer to next element
20. end if
21. end for
22. end for
23. if(Count > $Count_1$)
24. replace existing LCSS with new LCSS
25. set $Count_1$ to Count, set Count to zero
26. end if
27. remove L^{th} element of m and increment L by 1
28. set TP to zero
29. nd while

Proposed algorithm for LCSS

Proposed algorithm for SLCSS: Consider the time series given in section 4. SP stores the position of matched element in time series S where TP stores the position of 2^{nd} time series T. Count records the number of consecutive matches between two time series where C_1

stores maximum number of consecutive matches so far. MD is assigned a value which is equal to half the length of shorter time series. L is loop counter and class K_1 acts as a temporary storage for SLCSS's. If the length of SLCSS in class K_1 is greater than that of K_2, it replaces SLCSS of K_2 with that of class K_1. K_2 stores the SLCSS of maximum length.

1. while (L ≤ MD and K_2<length of m)
2. for i:L to m
3. for j:TP to n
4. if(S[i] = T[j] and (SP=0) and (TP=0)) then
5. add element to class K_1
6. set SP to i
7. set TP to j plus 1
8. increment Count by 1 and exit j loop
9. else if(S[i] ≠ T[j] and (SP=i+1) and (TP=j)) then
10. add element to class k
11. set SP to i and TP to j plus 1
12. increment Count by 1 and exit j loop
13. else if (S[i] ? T[j] and (Count>C_1)) then
14. set C_1 equal to Count
15. replace SLCSS in K_2 by K_1
16. set Count to zero
17. empty class K_1
18. end if
19. end for
20. end for
21. remove L^{th} element from m
22. increment L by 1 and set TP to zero
23. end while

Proposed algorithm for SLCSS

Results and Evaluations: In order to test and validate the pragmatic usefulness of proposed algorithms various simulations were performed. Both algorithms were implemented in C++ programming language. Under similar conditions, both algorithms [20] were executed and tested on same time series of different lengths. The running time comparison of proposed algorithm to that of dynamic programming based algorithm is presented in Fig. 2. It shows that the proposed algorithm requires less time to find LCSS than that of dynamic programming based algorithm. The performance of different algorithms starts decreasing when time series length is increasing. But the proposed algorithm performance is stable for shorter as well as longer time series. The efficiency of dynamic programming based algorithm is reducing with increase in time series length.

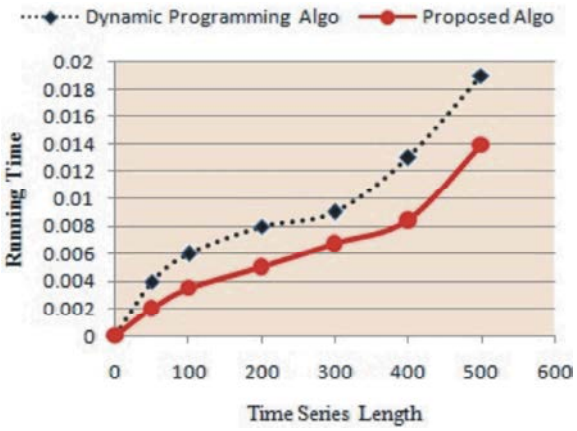

Fig. 2: Proposed algorithm running time vs dynamic programming based algorithm

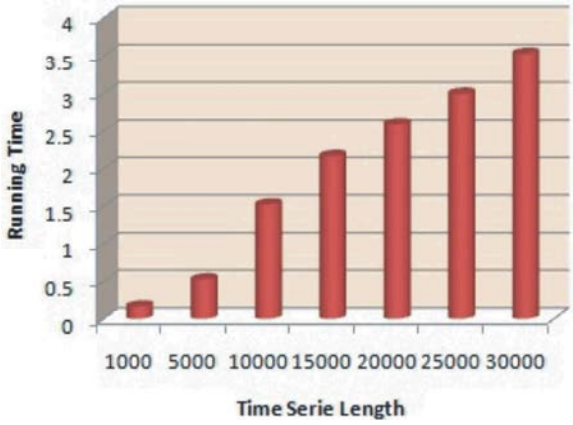

Fig. 3: Proposed Algorithm running time in seconds for longer time series

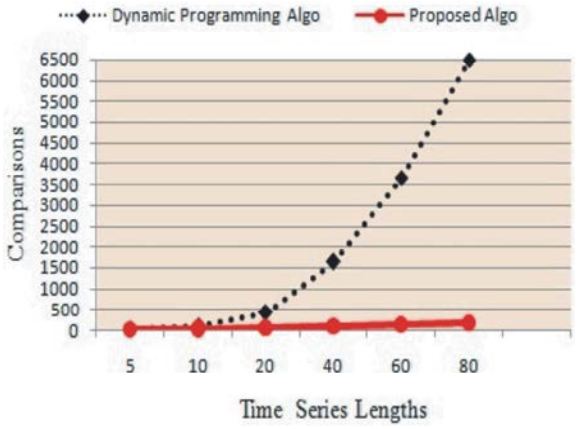

Fig. 4: Number of comparisons made by both algorithms for similar time series

Most of the non-metric based algorithms halt when the time series length crosses certain threshold values. The proposed algorithm running/execution time for longer

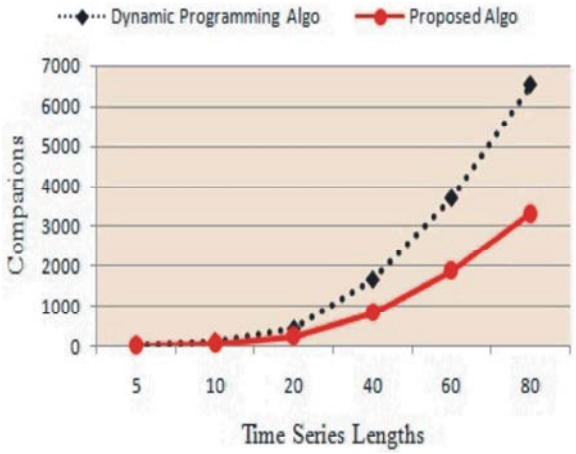

Fig. 5: Number of comparisons made by both algorithms for variable time series

time series is shown in Fig. 3. It is checked for time series of length 30,000*30,000 on real time data. The results show that its performance is better for longer time series as well.

In order to evaluate the performance of our algorithm in terms of the number of comparisons made for finding LCSS, we considered two different cases of time series. The first case is when two time series are exactly the same. Fig. 4 shows the number of symbols compared by proposed algorithm to that of dynamic programming based algorithm for first case. The proposed algorithm performs better than dynamic programming based algorithm in number of comparions made for finding LCSS. For case-1 interpretation, consider time series S_1=ABCDE and S_2= ABCDE. Dynamic programming algorithm makes 25 comparisons for filling out the matrix containing information about matched locations and additional 5 for printing it. The proposed algorithm makes 5 comparisons for calculating LCSS and 5 five more for printing it a total of 10 comparisons. 60% of unnecessary comparisons are avoided by the proposed algorithm. Similarly, for longer time serie 60% reduction in number of comparisons result in faster execution of proposed algorithm.

The second case is when two time series are of different lengths, different symbols at particular locations and having an LCSS of length less than or equal to the shorter time series length. The number of comparisons made by both algorithms is presented in Fig. 5. The proposed algorithm performance is far better than that of dynamic programming based algorithms. Consider the time series given in Fig. 1. The DPA makes 100+11 comparisons for calculating and printing the LCSS, whereas the proposed algorithm makes 46+8 comparisons for finding and printing the desired LCSS and results in more than 50% reduction in number of comparisons.

CONCLUSION

In this paper, we proposed an algorithm for finding similarity between time series. It calculates the longest common subsequence (LCSS) by avoiding unnecessary comparisons that reduces its performance. Its running time is far better than dynamic programming based algorithm and it is due to time control parameter δ. The proposed algorithm is tested on shorter as well as longer time series and shows good results for both cases. The space complexity is reduced by storing only matched symbols that are part of LCSS.

A sub/special type of LCSS called special longest common subsequence (SLCSS) is introduced. Its symbols appeared at consecutive location in both time series. A robust algorithm [19] is presented for the calculation of SLCSS. It will be used as an alternative to LCSS based decision support systems especially in agriculture, stock market and habitat monitoring. Our future direction includes development and implementation of a single algorithm for both LCSS and SLCSS.

REFERENCES

1. Mucherino, A., P.J. Papajorgji, P.M. Pardalos, 2009. Data mining in agriculture, 1st ed., illus, Springer Verlag.
2. Bergroth, L., H. Hakonen, H. Vaisanen, 2003. New Refinement Techniques for Longest Common Subsequence Algorithms, In String Processing and Information Re-trieval Lecture Notes in Computer Science, 28: 287-303.
3. Ding, H., G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh, 2008. Querying and Mining of time Series Data: Experimental Comparison of Representations and Distance Measures, Journal Proceedings of the VLDB Endowment, pp: 1542-1552.
4. Elzinga, C., S.B. Rahmann, H. Wang, 2008. Algorithms for subsequence combinatorics, Theoretical Computer Science, 409: 394-404.
5. Agrawal, A., K.I. Lin, H.S. Sawhney, K. Shim, 1995. Fast Similarity Search in the Presence of Noise, Scaling and Translation in Time-Series Databases. In: Proceedings of 21th International conferece on Very Large Data Bases, pp: 490-501.

6. Faloutsos, C., M. Ranganathan, Y. Manolopoulos, 1994. Fast Subsequence Matching in Time-Series Databases, In: Proceedings of SIGMOD Conference, pp: 419-429.
7. Agrawal, R., C. Faloutsos, A. Swami, 1993. Efficient similarity search in sequence databases, In: Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms, pp: 69-84.
8. Hsu, W.J. and M.W. Du, 1984. New Algorithms for the LCSS Problem, Journal of Computer and System Sciences, 29: 133-152.
9. Rick, C., 1995. A new flexible algorithm for the longest common subsequence problem, In: Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching Lecture Notes in Computer Science, pp: 340-351.
10. Wang, H., 2007. All Common Subsequences, In: M.M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International conference on Artificial Intelligence.
11. Wagner, R.A. and M.J. Fischer, 1974. The string to string correction problem, Journal of the Association for Computing Machinery, 21: 168-173.
12. Hirschberg, D.S., 1977. Algorithms for the Longest Common Subsequence problem, Journal of the Association for Computing Machinery, 24: 664-675.
13. Hun, J.W. and T.G. Szymanski, 1977. A Fast Algorithm for Computing Longest Common Subsequences, Communications of the ACM, 20: 350-353.
14. Mukhopadhyay, A., 1980. A Fast Algorithm for the Longest-Common-Subsequence Problem, Information Sciences, 20: 69-82.
15. Afroza, B., 2008. A Greedy Approach for Computing Longest Common Subsequences, Journal of Prime Research in Mathematics, 4: 165-170.
16. Nakatsu, N., Y. Kambayashi, S. Yajima, 1982. A Longest Common Subsequence Suitable for Similar Text Strings, Acta Informatica, 18: 171-179.
17. UCI KDD Archive [WWW Document], 2012. . URL http://kdd.ics.uci.edu/.
18. HOBO® U30 Remote Monitoring Systems - Live Data Feeds | Data Loggers: HOBO®Data Logger Products by Onset [WWW Document], 2012. URL http://www.onsetcomp.com/live_s
19. Zakarya, M. and A.A. Khan, 2012. Cloud QoS, High Availability and Service Security Issues with Solutions. IJCSNS, 12(7): 71.
20. Zakarya, M., A.A. Khan and H. Hussain, 2010. Grid High Availability and Service Security Issues with Solutions.
21. Zakarya, M., N. Dilawar, M.A. Khattak and M. Hayat, 2013. Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core. World Applied Sciences Journal, 22(10): 1431-1439.