Vol.2011-MPS-85 No.30
2011/9/16

スマートハウスのセンサデータに対する SAX を利用したイベント検出の検討

大西 史花* 渡辺 知恵美†

近年，ネットワークやセンサ技術が普及してきたことにより，様々なセンサが身の回りで使われるようになってきている．この動向に注目し，Ocha House プロジェクトでは，近未来における新しい住宅のかたちを提案している．我々は，住宅に設置されたセンサの情報に着目し，他のアプリケーションの基盤となるような，住宅の情報を格納し効率的にイベントが検出できる DB システムを提案する．我々は，高速な検索のために Symbolic Aggregate Approximation (SAX) を用いて時系列データの文字列化を行い，加えてランレングス圧縮，接尾辞木を適用し索引を作成する．本稿では，作成した SAX 索引を用いた，住宅内でのセンサデータからのイベント検出を行うための問合せ方法を検討する．

Event Detection using Archived Smart House Sensor Data obtained using Symbolic Aggregate Approximation

Ayaka ONISHI* Chiemi WATANABE †

In recent years, the widespread usage of network and sensor technologies has resulted in an increased number of applications that use various types of sensors. We propose the development of a futuristic house that can adapt to rapid changes in information technology. This project is called “The Ocha House Project.” We focused on data obtained using sensors and proposed and implemented a database system, which stores information pertaining to critical parameters in the house. This system can form the basis for developing other applications. We used the symbolic aggregate approximation (SAX) to quickly retrieve data. SAX converts data from a time series into a string. In addition, we created an index by applying suffix trees. In this study, we investigated queries that can be used for detecting events in a smart house using the SAX index and archived sensor data.

1. Introduction

Owing to the widespread use of networks and sensors technology in recent years, significant opportunities for using sensors have emerged places such as offices, and shops.

In this study, we focus on sensor applications in daily life. We are currently participating in the “Ocha House Project” which aims to develop a futuristic house that would facilitate a life style using a variety of IT technologies [1]. The methods and systems proposed in the project will be implemented in an experimental smart house called “Ocha House” (Figure 1) [2].

Figure 1. Experimental smart house called “Ocha House.”

In our study, we focus on technologies involving the storage of sensor data in a manner that enables the system to efficiently answer queries arising from using sensor applications. By querying sensor data, applications can extract “house’s information.” This includes information about events that occur in the house, such as the time of opening or closing a particular door, the duration for which the light in the living room is used, among others. If such information can be retrieved, many useful devices can then be implemented using the data. Some examples include tools that manage who may enter or exit a room and those that calculate the relationship between the behavior of occupants in the house for a given time period and the corresponding electricity bill.

To extract such data, sensor data needs to be archived into a database system. However, because sensors continuously measure parameters and transmit data, sizeable amounts of data will need to be stored on the database. To efficiently retrieve and analyze this data, we need to provide a methodology that can respond to queries for such large volumes of data.

*,† お茶の水女子大学

Thus, we propose a methodology that can effectively retrieve data on house parameters from the large dataset that has been archived. First, we leverage the symbolic aggregate approximation (SAX) index structure [3] proposed by Lin et. al.

The purpose of SAX is to quantize the original sensor data, which is usually a sequence of numerical values, and translate them to character strings. In this paper, we name the character strings according to the sensor data as "SAX index".

We assume that queries are issued using the query by example (QBE) style. Therefore, sequences of numerical values are issued as a query, and the system should then answer several numerical sequences, which are similar to the query pattern. Our system translates query sequences into the corresponding character string and retrieves substrings of SAX indices that are similar to the query string. One of the features of the SAX index is that various techniques dealing with character strings can be utilized for query retrieval and data mining. For example, motif discovery [4] using SAX can be implemented.

Because the sensor continually measures data and the sensor data sequence also increases in length, the corresponding SAX index will be a long string. Therefore, we apply the suffix tree as a method for effectively retrieving substrings from a long SAX index. However, we should note that the suffix tree index is usually used for exact matches. For event detection such as the opening and closing of doors, the system needs to perform similarity retrieval because sequences assigned to identical events have slightly different patterns depending on the person(s) triggering the event.

As a similarity measure, we then propose the application of similarity retrieval according to the edit distance. We define an editing cost for the edit distance based on the definition of the SAX approximation distance and are able to realize the fast similarity retrieval using the error-tolerant recognition algorithm [5], which deals with the edit distance retrieval method with a tree.

In Chapter 2, we explain the SAX method. Chapter 3 describes the application of the suffix tree to the SAX index, while chapter 4 presents the application of an index to the stored sensor data. In chapter 5, we describe the similar retrieval method using the edit distance, and Chapter 6 describes the investigations of event detections. Chapter 7 briefly discusses related research, while the conclusions and challenge faced are mentioned in chapter 8.

2. SAX

SAX is an expression technique used for time series data, and it quantizes this data into a string. Two parameters are required for generating a SAX index: length of the string *w* (or the amount of data for converting one alphabet) and the number of alphabetic types *a*. These values are experimentally determined on the basis of the data. The steps for converting time series data into a string are as follows: (Figure 2 shows an example, where the dashed line is the sensor data that applies SAX.)

(1) Data is divided into w equal sized "frames."
(2) The mean value of each frame is calculated.
(3) The regions with numerical sensor data values are divided into multiple sub-regions, and the alphabetic characters a, b, and c are assigned to the corresponding sub-regions.
(4) The mean value obtained in (2) is converted into characters according to the sub-region that is described in (3).

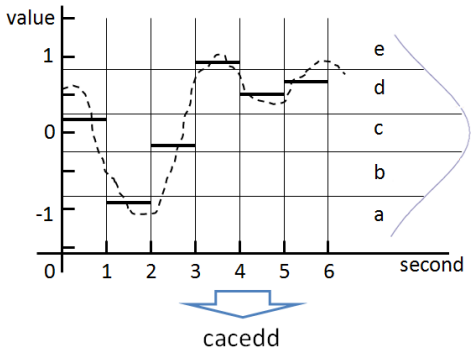


Figure 2. Time series data converted into a string by SAX.

There are three advantages of using SAX: (1) it is easy to implement, (2) it has a "symbolic" approach that allows lower bounding of the true distance, and (3) users can apply retrieval and data-mining techniques for obtaining the text string.

In this study, we implemented the methods described above and created a SAX index that converted sensor data into strings [1]. A query to the SAX index must first be converted to a string query. Time can be calculated from the position of the substring that matched the query.

We have introduced SAX as an indexing technique, and we translate sensor data into character strings.

In this system, we assume that queries are issued using the QBE style. That is, a sequence of numerical values is issued as a query, and the system responds to several sequences of numerical values that are similar to the query sequence. Our system also translates query sequences into the corresponding character string and retrieves substrings of the SAX index that are similar to the query string.

To perform a similarity, the system calculates the distance between the query sequence (Q) and subsequences of the sequence stored in the database. ({C1,…,Cn|Ci refers to the subsequences of the sequence C}) and answers a set of subsequences whose distances are less than a specified threshold. The Euclidean distance is often used as a measure of distance in time series data (Fig. 3 (A)).

On the other hand, the distance between a query string $\hat{Q}$, which is translated into a character string by SAX, and $\hat{C}_i$, which is one of the substrings of the SAX index, is defined as the sum of the square of the distance between dist($\hat{q}_i$, $\hat{c}_i$) (Fig. 3 (B)), where $\hat{q}_i$ and $\hat{c}_i$ are the *i*-th characters in the strings $\hat{Q}$ *and* $\hat{C}$, respectively, *w* is the number of characters in the SAX index, and *n* is the amount of original data.

$$(A)\quad \text{Original data}\quad D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

$$(B)\quad \text{SAX}\quad MINDIST(\hat{Q},\hat{C}) \equiv \sqrt{\frac{n}{w}}\sqrt{\sum_{i=1}^{w}(dist(\hat{q}_i,\hat{c}_i))^2}$$

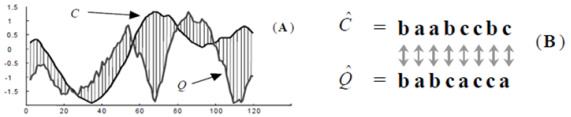
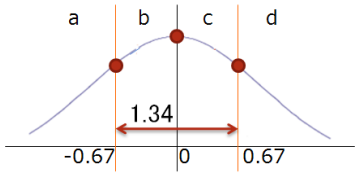


Figure 3. Definition and a schematic of distance between strings in SAX.

The distance dist($\hat{q}_i$, $\hat{c}_i$) between the characters $\hat{q}_i$ and $\hat{c}_i$ are defined by the distance between the maximum value assigned to the character $\hat{q}$ and the minimum value assigned to the character $\hat{c}$. For example, the left side of Figure 4 shows the distribution of values in the time series data. Since the SAX index normalizes values before quantization, they follow a normal distribution. The time series values are divided into four types of characters, as shown in Figure 4. The distance dist("a," "d") is found to be equal to 1.34 because the maximum value for "a" is −0.67, and the minimum value for "d" is 0.67. There is no need for the system to measure the distance between characters whenever a query is issued because a table showing the distances between characters can be prepared in advance (The table in Figure 4 shows the distance).

By defining these distances, the distance between the strings using SAX is the approximate distance for the original time series data and is guaranteed to allow lower bounding of the true distance.



	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

Figure 4. Table and illustration of the distance between the characters in SAX.

3. Application of suffix tree to SAX index

Queries using the SAX index retrieve substrings of the SAX index, which are similar to the query string. However, when the SAX index becomes a long sequence, it is not efficient to retrieve subsequences by sequential scanning. We therefore apply a suffix tree to the SAX index as a data structure that allows fast string retrieval.

The suffix tree is a data structure that represents suffixes of a given string as a tree. Each suffix corresponds to a different path in the tree. A leaf node takes the position of the corresponding suffix in the original string. For example, Figure 5 shows the suffix tree construction for the suffixes in the SAX index "babac." The suffix of the string "babac" is five: "babac," "abac," "bac," "ac," and "c." When users want to retrieve data, they will traverse a suffix tree beginning at the root. For example, if a user retrieves "ac" on the tree in Figure 5, he/she arrives at the leaf node 3. Therefore, he/she will know that "ac" exists in the third position from the beginning of the original string.

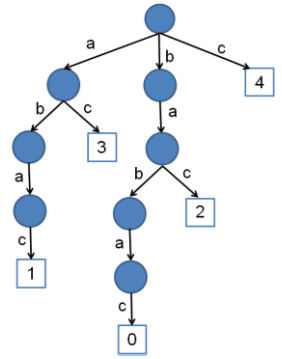


Figure 5. Suffix tree construction for the string "babac."

To confirm the effectiveness of the suffix tree, we compared the retrieval speed using the SAX index for data that had been archived over a two-day period and corresponded to 2,230,000 characters. Table 1 shows the results obtained, and they confirm that the SAX index that used the suffix tree retrieves data faster than a sequential scan.

Table 1. Comparison of speed of retrieval.

suffix tree	No suffix tree (Sequential Scan)
3ms	267ms

4. Applying indices to sensor data

Data are sent continuously by the sensor, and the end point of the sensor data sequence is always changing. Therefore, we should note the following two problems that may be encountered when the SAX index and suffix tree are generated for the sensor data.

(1) The sensor data should be normalized before generating the SAX index. However, we need to use the data's statistics for the normalization process. It is difficult to normalize the data correctly because sensor data are continuously being accumulated.
(2) The suffix tree is a tree of the suffix pattern. Since suffixes change every time new sensor data arrives, the depth of the tree increases exponentially.

To solve the first problem, we currently record the statistics when the accumulated data has reached an appropriate length. We currently set up the accumulated data for a one day. Sensor data is then normalized using these statistics. However, this approach is a temporary solution, and it is necessary for an alternative solution to be found. If the distribution of sensor data changes significantly after recording the statistics, there is no guarantee that the lower bounding of the approximate distance is allowed because the values assigned to letters are based on the normal distribution.
In order to resolve the second problem, we specified a limit for the depth of the suffix tree. This approach is similar to the use of sliding windows for retrieving subsequences of time series data. In SAX [1], the same approach for dimensionality reduction has been applied; hence, our approach is appropriate, except for the case where very long events are to be detected.

5. Preliminary investigations for event detection

In this section, we consider whether events that are categorized as "house information" can be detected using the SAX index, which applied a suffix tree. For example, the event "opening and closing a door" generates several sequence patterns. The speed of the moving door depends on the person who opens the door and/or the situation in which the door is opened. We need to examine whether such patterns can be returned by a query using the SAX index and the corresponding suffix tree index. However, we are already aware of two problems that exist when detecting events by query processing using a SAX index and suffix tree.

The first problem is that normal suffix trees cannot deal with similarity matches. SAX can perform a similarity match using an approximate distance to compare two strings. However, the suffix tree was applied to quickly retrieve substrings from a long SAX index. Because the suffix tree is essentially a data structure for substrings that match exactly, a similarity match cannot be performed. For example, Figure 6 shows the sequence data for the door sensor. From the sequence, it is observed that there are three behavior patterns. Subsequence (A) and (B) show the sensor value behavior when the same person opens and closes the door twice, while subsequence (C) indicates that the person who opened the door kept it open for a longer time period, after which it was closed. The bottom of Figure 6 shows the SAX index (w = 53, a = 8) that was converted into string values. For visualization purposes, the SAX index is compressed using run length encoding. If the retrieved query is a time sequence (Q), (A), (B), and (C) are not returned as the result because neither of them match the string (Q) exactly.

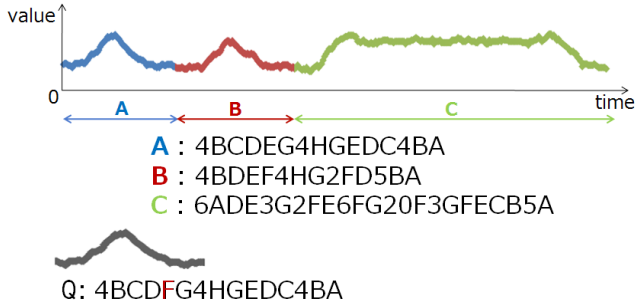


Figure 6. Behavior of the actual sensor data during opening and closing of door.

The second problem concerns the inability to determine the difference in speed when the door is being opened and closed. For example, Figure 7 shows the variation in the sensor data sequence as someone first opens and closes the door at normal speed, after which he/she slowly

opens and closes the door. The bottom of Figure 7 shows the SAX index (w = 41, a = 8) that was converted into string values.

The time series A and D represent the same event ("opening and closing the door"), but they have a slightly different SAX index owing to the difference in the speeds with which the actions were performed. Currently, we cannot differentiate between these values.

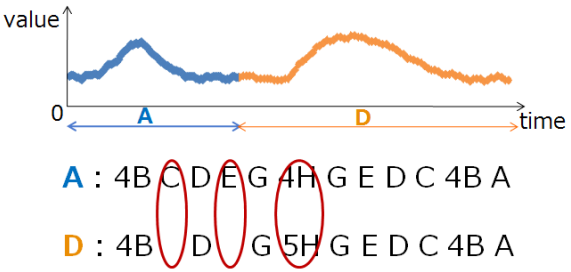


Figure 7. Difference in opening and closing time of door.

6. Similar retrieval method using the edit distance

In order to facilitate situations such as those mentioned in section 5, we define the edit distance as the distance between the strings.

The edit distance is the minimum total value of the costs (replace, delete, insert) required for converting a string Q into a string C. The minimum total value is considered to be the distances dist(Q,C) between the string C and the string Q. For example, if the string Q is "AFC" and the string C is "ABCE," strings Q and C will match if the second character of Q (character "F") is replaced by the character "B" and if the character "E" is inserted into the fourth character of Q. If the cost is defined as the number of "replace" and "insert" operations that are required, then, dist(Q,C) will be 2.

An approach that can be used to retrieve a partially similar string using a suffix tree and applying the edit distance has been proposed as the error-tolerant recognition algorithm [3]. This can be adopted to perform similar retrieval of strings using a suffix tree.

The second problem described in the previous section may be solved using the dynamic time warping (DTW) distance.

DTW is a measure of the distance of the time series. It can be calculated using the distance absorbed scaling and the time gap. Chen [5] describes the application of DTW to the cost value of the edit distance. We can therefore calculate the edit distance based on DTW by applying this approach.

In section 6.1, we describe the edit distance in the SAX index, and in section 6.2, we describe the experiments involving the detection of the "doors" event.

(1) Definition of edit distance in SAX index

According to DTW, the editing costs of the editing distance comprise.

- Replacement Cost

  The replacement cost is determined by the distance dist($\hat{q}$, $\hat{c}$) between characters in SAX (section 2).

- Gap (insert and delete) Costs

  The insert cost is defined as the distance between the inserted character x and the character y, which is before one of the positions into which the insertion is made. For example, to compare the costs of "ABC" and "AC," the insert cost associated with inserting B between A and C in the second string is dist(A,B). The same concept applies to the delete cost, which is defined as the distance between the delete character x and the character y, which is before one of the positions that is to be deleted.

Based on this concept, the distance of a string from the first character to the respective i-th and j-th characters of Q and C in the SAX index can be defined as follows:

$$P(Q_i,C_j) = \min\{P(Q_{i-1},C_{j-1})+dist(q_i,c_j), \quad \#permute$$
$$P(Q_{i-1},C_j)+dist(q_{i-1},c_j), \quad \#delete$$
$$P(Q_i,C_{j-1})+dist(q_i,c_{j-1})\} \quad \#insert$$

However, in our case, we anticipate that user requests will include the desire to control the speed with which a door is opened or closed. Therefore, in order to consider speed, the retrieval uses the following equation:

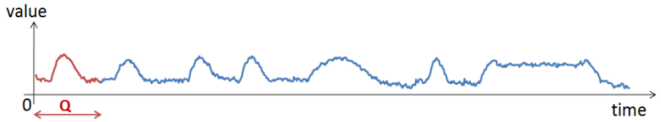
$$P(Q_i,C_j) = \min\{P(Q_{i-1},C_{j-1}) + dist(q_i,c_j), \quad \#permute$$
$$P(Q_{i-1},C_j) + \alpha*dist(q_{i-1},c_j), \quad \#delete$$
$$P(Q_i,C_{j-1}) + \alpha*dist(q_i,c_{j-1})\} \quad \#insert$$

A control parameter α can be specified by a user on delete and insert cost. If α is increased, the retrieval can include speed information for the "doors" event.

(2) **Experiments involving "doors" event detection**

We performed "doors" event detection experiments using the edit cost as it was defined in the previous section. Figure 8 shows SAX strings of the experimental data and a query Q. The seven events that are observed include four successive open-and-close events, followed by a slow open-and-close event and a fast open-and-close event, and finally, a case where a subject kept the door open for a longer time before closing it. For this example, we performed detection experiments during the time period shown in red as the query Q.

Figure 9 shows the results of detection experiments that resulted in changed thresholds, delete, and insert costs. The orange parts are substrings of the detection result.



Q:C2BCFH7JIHGFD2C

Figure 8. Behavior of experimental data（w=287, a=10）

The threshold in case (1) is set to 0.2 and α = 1, while the threshold in case (2) is set to 0.2 and α = 10.

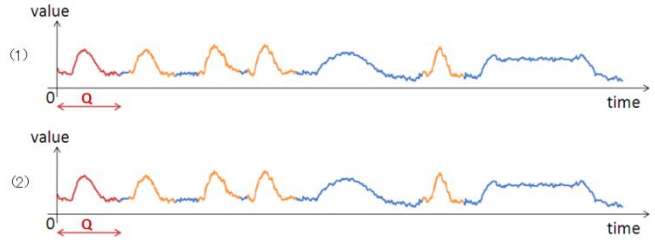


Figure 9. Detection results

In case (1), the first three events and the faster event could be viewed as detection results. However, when only α is varied and the threshold is not varied as in case (2), neither the fifth event nor the last event were detected. We had expected that all events would have been detected in case (2), but the result in case (2) were the same as in case (1), the parameter α did not function correctly. One possible reason may be that it is too small to affect the delete cost. Therefore, we will not regulate the collective costs to insert and delete, but will instead separately regulate them.

7. Related research

SAX is based on the piecewise aggregate approximation [6], which quantizes time series data using a fixed interval; it is an extended approach that allows the lower bounding of the true distance in the approximate distance. It can be applied to a variety of search technologies and techniques for mining strings.

In this paper, we performed fast retrieval of partial time series data using suffix trees and edit distances. Chen also performed a similar retrieval for the time series data using edit distance [7][8]. In [7], the original value was not quantized, and the cost was determined on the basis of whether the distance between the two characters in the time series was within a specified threshold. In [8], a cost is defined that satisfies the axioms of distance and speeds up the retrieval using the measured index.

In addition, iSAX is an extension of SAX and is applicable for very long time series [9]. This approach generates an index for short SAX strings that are split using a sliding window and has achieved a fast retrieval for substrings of a long string. Unlike the suffix tree that we adopted, the iSAX index considers n-dimensional data as a string of n characters that are divided using a sliding window and which generate an index on the feature space using an n-dimensional vector. However, this approach does not correspond to the scaling of time series.

Gao et. al. [10], Zhi et. al. [11], and others have conducted research into part similarity retrieval of streaming data. Sakurai et. al have studied a similar retrieval method using high-speed DTW in streaming data [12].

We have tried a similar retrieval method using a speed-up approach of the string retrieval method for time series data that is quantized by SAX. In future, the effectiveness of these methods needs to be verified.

8. Conclusions

In this paper, we implemented a SAX index that allows fast retrieval of archived sensor data for a smart house. In addition, we achieved fast similarity event detections using a suffix tree and performed experiments for event detections.

In the future, we will focus on improving the proposed retrieval method and perform additional experiments using different events.

Reference

1) Ocha House:
http://www.siio.jp/index.php?OchaHouse
2) Ocha House Projects:
http://www.siio.jp/index.php?OchaHouseProjects
3) Jessica Lin , Eamonn Keogh , Stefano Lonardi , and Bill Chiu. : "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms," SIGMOD workshop, 2003.
4) Yoshiki Tanaka and Kuniaki Uehara: "Motif Discovery Algorithm from Motion Data," 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI), June 2004.
5) Kemal Oflazer: "Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction," Journal of Computational Linguistics, Vol. 22, Issue 1, March 1996.
6) Yi, B, K, and Faloutsos, C: "Fast Time Sequence Indexing for Arbitrary Lp Norms," 26st International Conference on Very Large Databases, pp 385–pp 394, 2000.
7) Chen L. Chen, M.T.zsu, and V. Oria: "Robust and efficient similarity search for moving object trajectories," CS Tech. Report. CS-2003-30, School of Computer Science, University of Waterloo.
8) Lei Chen and Raymond Ng: "On the marriage of Lp-norms and edit distance," 30th International Conference on Very Large Data Bases (VLDB 2004), 2004.
9) Jin Shieh and Eamonn Keogh: "iSAX: Indexing and mining terabyte sized time series," 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 2008.
10) Like Gao and Sean. X. Wang: "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series," The 2002 ACM SIGMOD International Conference on Management of Data, pp 370−pp 381, June 2002.
11) Y. Zhu and D. Shasha: "StatStream: Statistical monitoring of thousands of data streams in real time," 28th International Conference on Very Large Data Bases (VLDB 2002), pp 358–pp 369, August 2002.
12) Y. Sakurai, C. Faloutsos, and M. Yamamuro: "Stream Monitoring under the Time Warping Distance," 23rd IEEE International Conference on Data Engineering (ICDE 2007), pp 1046–pp 1055, April 2007.