An Improvement of Controlling Quality of Large Scale Water-Level Data in Thailand

Nuttapon Pattanavijit, Peerapon Vateekul
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Phayathai Road, Pathumwan
Bangkok, Thailand, 10330
Email: nuttapon.p@student.chula.ac.th, peerapon.v@chula.ac.th

Kanoksri Sarinnapakorn
Hydro and Agro Informatics Institute
Ministry of Science and Technology
Thailand, 10400
Email: kanoksri@haii.or.th

***Abstract*—Extremely change in precipitation level such as water level can cause severe damage. In order to acknowledge changes, Hydro and Agro Informatics Institute has installed telemetry system across Thailand to collect and analyze precipitation level. However, to use its data in researching, incorrect data must be filtered. In previous work, we successfully detect various problems in water level data accurately. Still, outliers detection algorithm and missing pattern algorithm proposed in the previous work still room for improvement in terms of running time and ease of implementation. One of problems is that both algorithms is relied on complicated clustering which is unnecessary. In this paper, we propose to improve the clustering algorithm used in the previous work for water-level data. A linear clustering algorithm is invented and used to replace the old clustering algorithm. As a result, we can speed up the outliers detection algorithm and hold the same running time on missing pattern algorithm but increase simplicity and ease of implementation. Moreover, compared with the previous work we measure actual running time of our algorithms and found that they significantly help reduce the running time.**

***Index Terms*—Do, not, forget, to, insert, keyword, here**

I. Introduction

Thailand has faced many dreadful climate-related disaster including floods, droughts, and tropical cyclones. They have happened year after year and cause severe loss. For example, In 2011, seasonal flooding results in US$ 45.7 billion damage to Thailands economy, which is 1.1% of the countrys GDP [1]. Also, the climate-related disaster bring difficulties to agriculture and industry, which are backbones of Thailand economy. To handle these disaster, the government established many organization to cooperatively counteract with it. One of the organization is Hydro and Agro Informatics Institute (HAII). HAII's main focus is to research and utilize knowledge in agricultural and water resource management to confront the climate disaster [2].

In order to conduct researches, they have collected precipitation data by installing telemetry systems across Thailand. The telemetry system is a device that is used to collect physical, chemical, and biological data from its various sensors [3]. For instance, rivers water level, rainfall level, humidity, and temperature can be measured. Currently, Hydro and Agro Informatics Institute (HAII) have already installed over 800 telemetry system across Thailand. Fig. 1 shows example of location where telemetry systems were installed. Each system send data from its many sensors back to central database every 10 minutes via cellular network. And, all of the data is stored order by timestamp. From these numbers, we can estimate that there are 3.45 million records of data added to database every month. Since HAII have collected precipitation data for over 5 years, we can assume that we are dealing with approximately 207 million records.

Sometimes, incorrect data is reported from the telemetry systems such as minus value of cumulative rainfall level, or rivers water level suddenly changed from one level to another, which are not possible. In addition, data loss can also be occurred due to poor cellular network in rural area. These inconsistent data is not suitable to be used in researches since it might lead to inaccurate results.

To filtered out incorrect data efficiently, our previous work [5] proposed algorithms to detect anomalies in water level data, which includes outliers, and missing pattern algorithm. Example of anomaly data detected by these algorithms is illustrated in Fig. 2, and Fig. 3. However, both outliers detection algorithms and missing pattern is heavily relied on clustering algorithm. *DBSCAN* algorithm is used to do the clustering task. The *DBSCAN* itself have $O(n \log n)$ running time if it is implemented using special data structures that support region query such as *R*-Tree* or *k-d Tree*, which seem to be very complex, and it have $O(n^2)$ running time if no special data is used. This can cause outlier detection algorithm's running time rise up to $O(n \log n)$ or $O(n^2)$, which is not quite suitable for over 200 million records of data. Moreover it unnecessarily add complication and time used in implementing both algorithms. Also, real runtime might be slow due to overhead of the algorithms.

In this paper, we proposed to improve the clustering algorithm in the previous work for water-level data. A linear clustering algorithm is invented and used to replace *DBSCAN* clustering. The linear clustering algorithm helps improve both outliers detection algorithm and missing pattern algorithm for water-level data. As a result, the linear clustering algorithm helps increase speed and simplicity, yet maintain same accuracy of the algorithms. The outliers detection algorithm's running time is dwindled to $O(n)$. The missing pattern algorithm

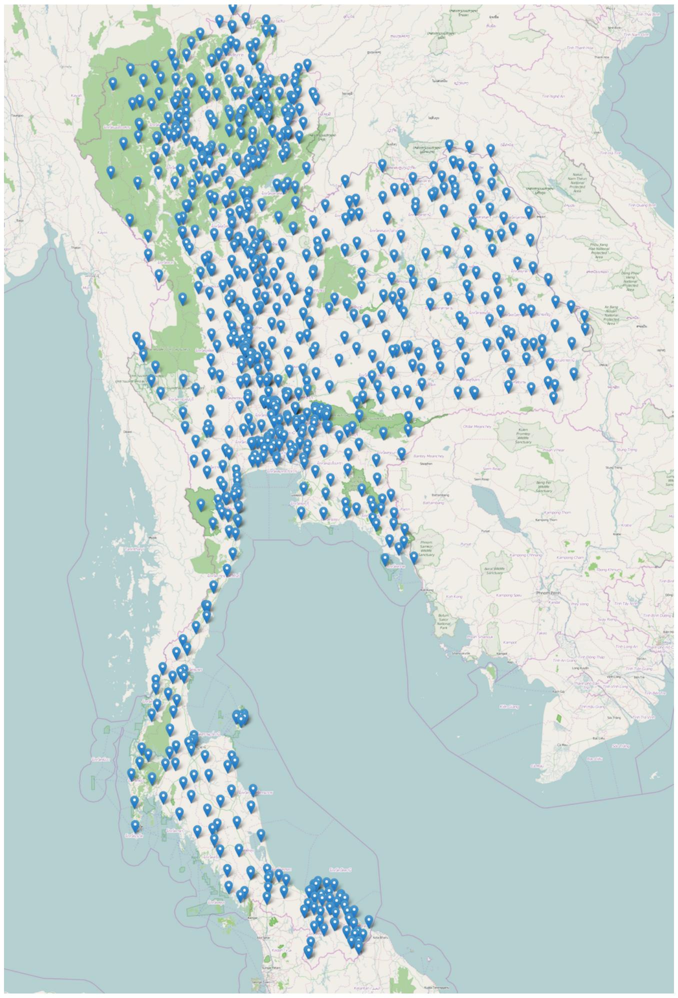
Fig. 1. Map of Thailand indicates where telemetry system was installed.

also achieves faster real running-time, even though holds the same theoretical running-time as previous work at $O(n \log n)$.

The paper is organized as follows. Section II recaps our previous work and points out its running time. Our approach to refine the both algorithms are illustrated in Section III. Experiment results are shown in Section IV. Last, Section V delivers a conclusion.

II. Previous Data Quality Management and Its Running Time

Our previous work proposed two algorithms to control quality of water-level data, outliers and missing pattern. Both of the algorithms apply clustering technique in order to detect anomaly data and pattern of data. DBSCAN was selected as a clustering algorithms. Note that for the time dimension, 10 minutes will be counted as a distance of 1 unit because data points is captured in 10 minutes interval and dataset is already sorted by timestamp.

In each subsection, since our previous previous work has not stated about the running time yet, we will brief each algorithm, illustrate its pseudocode, and analyze its running time in order to later refine both algorithms.

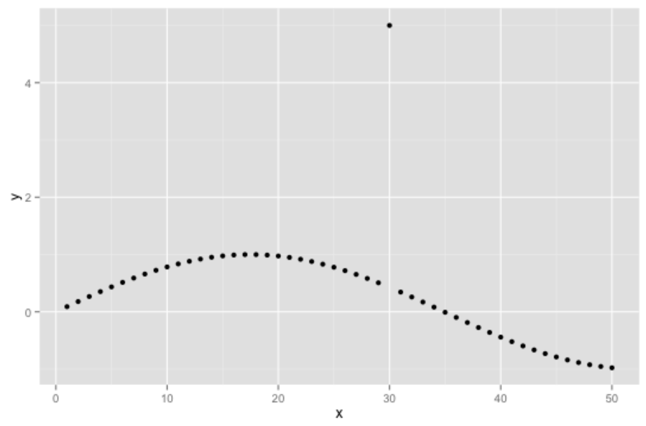

Fig. 2. Example of data with outliers.

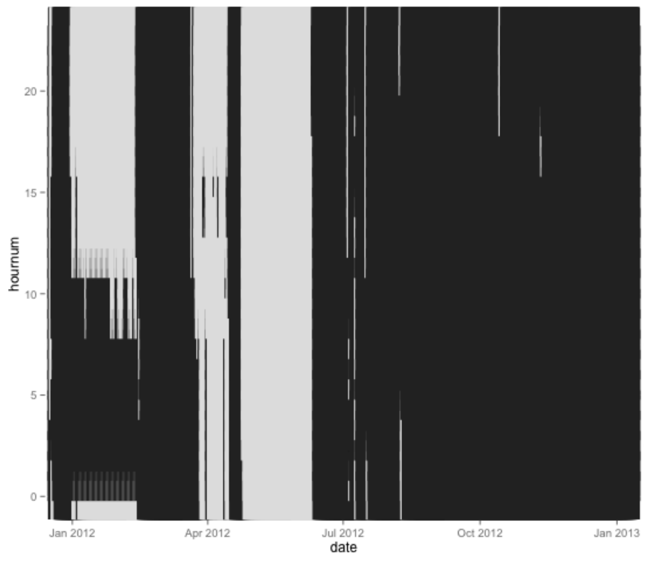

Fig. 3. Example of data with missing pattern. X-axis represent date of the data. Y-axis indicate hour of day of the data. The lighter area shows date and hour where data was missing.

A. Outliers Detection Algorithm

In previous work, we detect outliers by directly applies *DBSCAN* to water-level data. *DBSCAN*'s required parameters, minimum number of points *minpts* and distance epsilon *eps*, is set to 3 and 1.05 respectively in order to classify data which have extreme difference in value compared to adjacent data as a noise. Fig. 4 illustrates pseudocode of outliers detection algorithm. C is an array of cluster index. Data point d_i is in k-th cluster if $C_i = k$ and $k \neq 0$. If $k = 0$, d_i is considered as a noise. N is an array storing index of noise data. If $C_i = 0$, then $i \in N$.

From pseudocode in Fig. 4, it is obvious that running time of Detect-Outliers is based on DBSCAN. Thus, we can conclude that Detect-Outliers complexity is $O(n^2)$ or $O(n \log n)$ – depends on the data structure inside DBSCAN.

```
1: procedure DETECT-OUTLIERS($d$)
2:     $minpts \leftarrow 3$
3:     $eps \leftarrow 1.05$
4:     $C \leftarrow \emptyset$                ▷ Set of cluster of data index
5:     $N \leftarrow \emptyset$                     ▷ Set of noise data index
6:     $C, N \leftarrow$ DBSCAN($d, minpts, eps$)
7:     return $N$
8: end procedure
```

Fig. 4. Pseudocode of outliers detection algorithm.

```
1: procedure MISSING-PATTERN($d$)
2:     if $end - start \leq minDataReq$ then
3:         return $\emptyset$                          ▷ Too few data
4:     end if
5:     $f \leftarrow$ HOURLY-MISSING-FREQUENCY($d$)
6:     $minpts \leftarrow 1$
7:     $eps \leftarrow \frac{|d|}{2}$
8:     $C, N \leftarrow$ DBSCAN($f, minpts, eps$)
9:     $C' \leftarrow$ array of all $C_i \neq 0$
10:    $haveOverallMP \leftarrow$ UNIQUE($C'$) $\geq 2$        ▷ at least 2 clusters
11:
12:    $P_l \leftarrow$ MISSING-PATTERN($d_1 \dots d_{\frac{|d|}{2}}$)
13:    $P_r \leftarrow$ MISSING-PATTERN($d_{\frac{|d|}{2}+1} \dots d_{|d|}$)
14:    $P \leftarrow$ COMBINE-RESULT($P_l, P_r, haveOverallPattern$)
15:
16:    $mergeGap \leftarrow 15$                          ▷ 15 days
17:    return MERGE-OVERLAP-PATTERN($P, mergeGap$)
18: end procedure
```

Fig. 5. Pseudocode of missing pattern algorithm.

B. Missing Pattern Algorithm

Proposed in previous work, missing pattern algorithm is used to detect pattern of missing data. There are many steps required in missing pattern algorithm. First, it convert water-level data into frequency domain, which count how many data is missing during each hour ($f_0, f_1, \dots, f_{23}$). Second, we use DBSCAN to analyze the frequency. If there are more than one cluster detected, there must be at least two hours having significantly different missing frequency (f_i). Then, we determine whether it contains missing pattern in overall data. Next, divided-and-conquer technique was applied to find missing pattern in subset of data. Last, we merge overlapped missing result together.

To be easier to explain, we rewrite its pseudocode from previous work in figure 5. Result of MISSING-PATTERN procedure is returned as a list of tuples indicating start date and end date of pattern. Also, list is always sorted by start date of tuples.

From Fig. 5, HOURLY-MISSING-FREQUENCY on line 5 can be done in $O(|d|)$ by using counting sort with 24 bucket representing missing frequency of each hour of day ($f_0, \dots, f_{23}$). COMBINE-RESULT on line 14 is done in $O(1)$ by checking 8 possible cases. And, MERGE-OVERLAP-PATTERN could be executed in $O(|d|)$. Since the tuples in P is sorted and not literally overlap, we can sequentially check each adjacent pair of tuples whether it should be merged. For the *DBSCAN* in line 8, since f always have 24 elements, the *DBSCAN* always has $O(1)$ running time.

Since it uses divide-and-conquer, We can compute the complexity of by using *master method*, which the complexity of algorithm can be represent in:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

For missing pattern algorithm, $f(n)$ is equal to $O(n)$, which is the slowest running time in each recurrence. By applying *master method*, running time of the algorithm is:

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + O(n \log n) \\ &= O(n \log n) \end{aligned}$$

Even though the clustering algorithm is already has $O(1)$ running time, we think that using *DBSCAN* for only 24 data points is unnecessary and contain huge overhead. Also, if we need to implement the *DBSCAN* ourselves, it is too complicated to be use with only 24 data points.

III. PROPOSED IMPROVED ALGORITHMS

Our proposed algorithms aim to solve the bottleneck in outliers detection algorithm and simplify the clustering in missing pattern algorithm. To do that, we try to create a new clustering algorithm which can imitate *DBSCAN*'s result but with more efficiency, by using some facts about precipitation data and the algorithms themselves.

First, since the data was captured from a sensor, we can assume that at time t, there will be only one d_t. This fact implies that there is no need to search for adjacent points in two dimension.

Second, outliers algorithm and missing pattern algorithm produce best result when *minpts* $= 3$ and *minpts* $= 1$ respectively. Let C be an array of cluster index. Data point d_i is in k-th cluster if $C_i = k$ and $k \neq 0$. If $k = 0$, d_i is considered as a noise. We can say that if *minpts* ≤ 3, following properties are always true:

1) if $C_i = k; k \neq 0$ (k-th cluster) and $|d_{i+1} - d_i| \leq$ *eps*, then $C_{i+1} = k$ too.
2) $|\{i \mid C_i = k\}| \geq$ *minpts* for all $k \geq 1$

The first property shows that it is suffice to determine the cluster by measure distance between adjacent pair of data points. It is true because if *minpts* $= 1$, at least d_i can form its own cluster and add d_{i+1} to its cluster. If *minpts* $= 2$ and $C_i = k$, there must be another data point j where $C_j = k, j < i$ within distance *eps*. So i can be add freely into k-th cluster ($C_i = k$). Last, if *minpts* $= 3$, since i is already in C_k, there must be another data point j where $C_j = k, j < i$ within distance *eps*. So, if $|d_{i+1} - d_i| \leq$ *eps*, d_i will satisfy *minpts* $= 3$ (including itself) condition. Thus, $i + 1$ can be added into k-th cluster.

```
1: procedure LINEAR-CLUSTER($d, minpts, eps$)
2:     $C_{tmp} \leftarrow \emptyset$
3:     $C \leftarrow$ array of size $|d|$
4:     $k \leftarrow 0$
5:     $N \leftarrow \emptyset$
6:     for $i \leftarrow 1$ to $|d|+1$ do
7:         $newCluster \leftarrow$ false
8:         if $i = 1$ or $i = |d|+1$ or $|d_i - d_{i-1}| > eps$ then
9:             $newCluster \leftarrow$ true
10:        end if
11:        if $newCluster$ and $|C_{tmp}| \geq minpts$ then
12:            $k \leftarrow k + 1$
13:            $C_i \leftarrow k$ ; $\forall i \in C_{tmp}$
14:            $C_{tmp} \leftarrow \emptyset$
15:        else if $newCluster$ then
16:            $C_i \leftarrow 0$ ; $\forall i \in C_{tmp}$
17:            $N \leftarrow N \cup C_{tmp}$
18:            $C_{tmp} \leftarrow \emptyset$
19:        end if
20:        add $i$ to $C_{tmp}$
21:    end for
22:    return $C, N$
23: end procedure
```

Fig. 6. Pseudocode of new clustering algorithm.

The second property shows that every cluster must have at least *minpts* data points. Assume that cluster k is the smallest cluster possible, cluster k must have at least one core data point in order to form a cluster. Let d_i be the only core data point where $C_i = k$. d_i must have at least *minpts* data point within distance *eps* (including itself). So, we can conclude that $|\{i \mid C_i = k\}| \geq$ *minpts*.

With this two properties, we can create an algorithm which can imitate *DBSCAN*'s result for *minpts* ≤ 3. Fig. 6 illustrates its pseudocode.

From Fig. 6, the pseudocode work by following steps. First, it creates a temporary cluster an assume that d_i is in cluster (start from d_1). Then, it checks the distance to its next adjacent data point (d_{i+1}). If the distance is less than eps, put d_{i+1} in the same temporary cluster. Otherwise, create new temporary cluster and assume that d_{i+1} is in it. Before creating new temporary cluster, we determine whether the previous temporary cluster can be formed into real cluster by counting data points inside it. If it has more than $minpts$ points, it can be formed as a real cluster. If not, all data point inside it should be considered as noise.

The LINEAR-CLUSTER algorithm has $O(n)$ running time even though it contains nested loop in line 6 and line 13 or 16. Since every i will be a member of C_{tmp} only once, line 13 and 16 will be executed only n times in total. Thus, we can conclude that it has $O(n)$ running time.

Therefore, we can replace *DBSCAN* in outliers algorithm and missing pattern algorithm with LINEAR-CLUSTER to reduce it's overall complexity. The outliers algorithm become $O(n)$ in complexity. For missing pattern, although the clustering process already has $O(1)$ running time, but LINEAR-CLUSTER is far more uncomplicated compare to DBSCAN. This helps reduce overhead in real running time and decrease implementation time for developers.

IV. EXPERIMENTS

Lorem Ipsum

V. CONCLUSION

Lorem Ipsum

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.