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 detection algorithm proposed in the previous work still have room for improvement in terms of running time and ease of implementation. One of problems is that both algorithms rely complicated clustering which is unnecessary. In this paper, we propose the improvement of 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 detection 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 linear clustering algorithm significantly help reduce the running time for outliers detection algorithm.**

***Index Terms*—Data quality control, outlier detection, missing pattern detection, data improvement, clustering**

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 Thailand's economy, which is 1.1% of the country's GDP [1]. Also, the climate-related disaster bring difficulties to agriculture and industry, which are backbones of Thailand's 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, river's water level, rainfall level, humidity, and temperature can be measured. Currently, Hydro and Agro Informatics Institute (HAII) has already installed over 800 telemetry systems across Thailand. Fig. 1 shows localtion of installed telemetry systems. 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 has collected precipitation data for over 5 years, we can assume that we are dealing with approximately 207 million records.

Sometimes, incorrect data are reported from the telemetry systems such as a minus value of cumulative rainfall level, or an instantaneous change of river's water level, 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 [4] proposed algorithms to detect anomalies in water level data, which includes outliers detection algorithm, and missing pattern detection algorithm. Example of anomaly data detected by these algorithms is illustrated in Fig. 2, and Fig. 3. However, both outliers detection algorithm and missing pattern detection algorithm is heavily relied on clustering algorithm. *DBSCAN* algorithm [5] is used to do the clustering task. The *DBSCAN* itself has $O(n \log n)$ running time if it is implemented using data structures that support two dimensional region query such as *R*-Tree* [6] or *k-d Tree* [7], which seem to be very complex, and it has $O(n^2)$ running time if no sophisticated data structures is used. This causes 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 adds complication and time used in implementing both 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 detection algorithm for water-level data. As a result, the linear clustering algorithm helps increase speed and simplicity, yet maintain

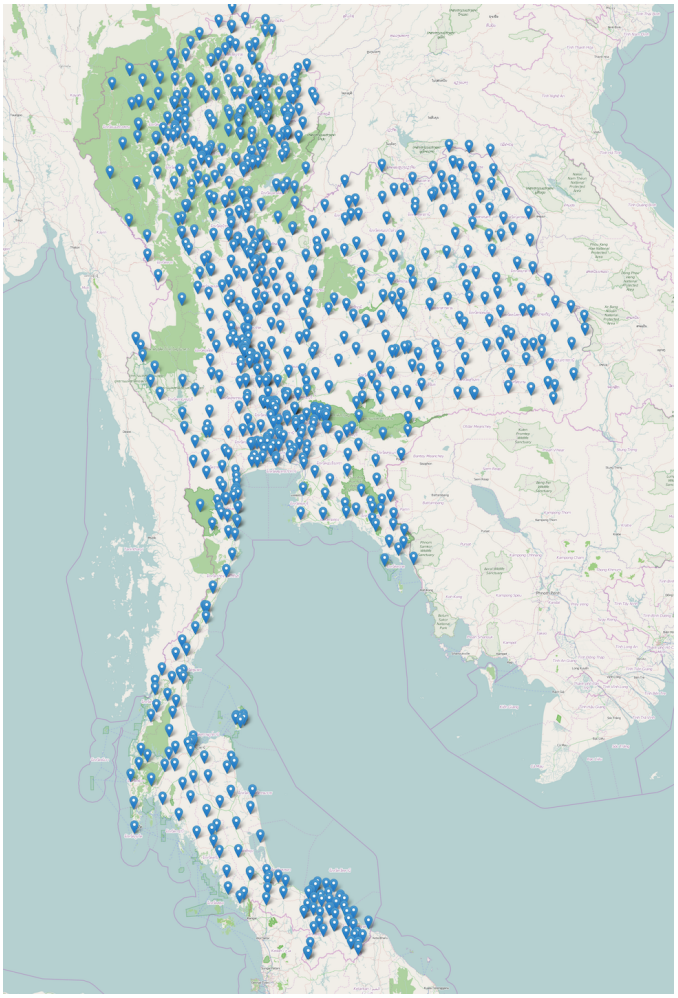

Fig. 1. Map of Thailand indicates where telemetry system was installed.

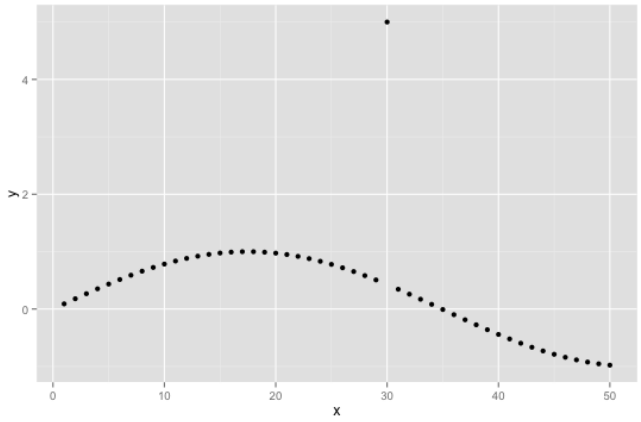

Fig. 2. Example of data with outliers.

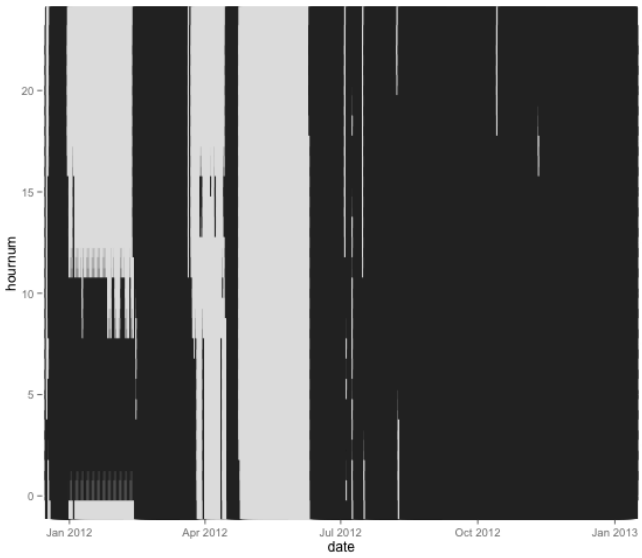

Fig. 3. Example of data with missing pattern. X-axis represent date of the data. Y-axis indicate hour of day of the data. The lighter area shows date and hour where data was missing.

same accuracy of the algorithms. The outliers detection algorithm's running time is dwindled to $O(n)$. The missing pattern detection algorithm also holds the same 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 using linear clustering algorithm 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 detection algorithm and missing pattern detection algorithm. 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 are captured in 10 minutes interval and data are 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.

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 a 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 $i \in N$.

```
1: procedure DETECT-OUTLIERS(d)
2:     minpts ← 3
3:     eps ← 1.05
4:     C ← ∅
5:     N ← ∅
6:     C, N ←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 ≤ minDataReq then
3:         return ∅                                  ▷ Too few data
4:     end if
5:     f ← HOURLY-MISSING-FREQUENCY(d)
6:     minpts ← 1                                    ▷ no noise
7:     eps ← 0.5·MAX-FREQ-POSSIBLE(d)
8:     C, N ← DBSCAN(f, minpts, eps)
9:     haveOverallMP ← |UNIQUE(C)| ≥ 2               ▷ at least 2
10:
11:    P_l ←MISSING-PATTERN(d_1 ... d_{⌊|d|/2⌋})
12:    P_r ← MISSING-PATTERN(d_{⌊|d|/2⌋+1} ... d_{|d|})
13:    P ← COMBINE-RESULT(P_l, P_r, haveOverallPattern)
14:
15:    mergeGap ← 15                                 ▷ 15 days
16:    return MERGE-OVERLAP-PATTERN(P, mergeGap)
17: end procedure
```

Fig. 5. Pseudocode of missing pattern detection algorithm.

From the 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.

B. missing pattern detection algorithm

Proposed in previous work, missing pattern detection algorithm is used to detect pattern of missing data. There are many steps required in missing pattern detection algorithm. First, it converts water-level data into frequency domain, which count how many data is missing during each hour $(f_0, f_1, \ldots, f_{23})$. If there is at least one data point in hour i of some particular day, that hour i of the day will not be considered as missing. 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 is applied to find missing pattern in subset of data. Last, we merge overlapped missing result together.

To make it easier to explain, we rewrite its pseudocode from previous work in Fig. 5. Results of MISSING-PATTERN procedure are 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 $(f_0, \ldots, f_{23})$. 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 by using *master method* [8], which the complexity of algorithm can be represent in:

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

For missing pattern detection 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:

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

Even though the clustering algorithm is already has $O(1)$ running time, we think that using *DBSCAN* for only 24 data points is unnecessary. 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 algorithm aims to solve the bottleneck in outliers detection algorithm and simplify the clustering in missing pattern detection 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 dimensions.

Second, outliers detection algorithm and missing pattern detection 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 $C_j = k, j < i$

```
1: procedure LINEAR-CLUSTER(d, minpts, eps)
2:     C_tmp ← ∅
3:     C ← array of size |d|
4:     k ← 0
5:     N ← ∅
6:     for i ← 1 to |d| + 1 do
7:         newCluster ← false
8:         if i = 1 or i = |d| + 1 or |d_i − d_{i−1}| > eps then
9:             newCluster ← true
10:        end if
11:        if newCluster and |C_tmp| ≥ minpts then
12:            k ← k + 1
13:            C_i ← k ; ∀i ∈ C_tmp
14:            C_tmp ← ∅
15:        else if newCluster then
16:            C_i ← 0 ; ∀i ∈ C_tmp
17:            N ← N ∪ C_tmp
18:            C_tmp ← ∅
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.

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.

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 is 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.

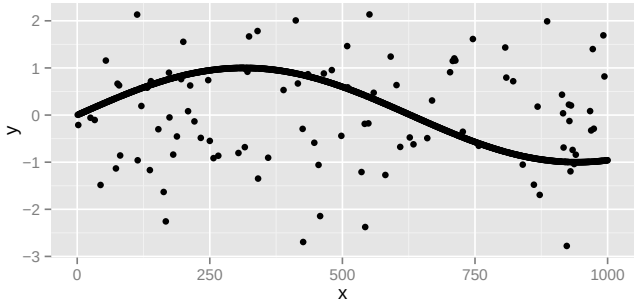

Fig. 7. Example of generated data (n = 1000) for first experiment.

Therefore, we can replace *DBSCAN* in outliers detection algorithm and missing pattern detection 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 decrease implementation time for developers.

IV. EXPERIMENTS

We conduct two tests. First, we compare the DBSCAN and LINEAR-CLUSTER directly in order to reflect the running time of outliers detection algorithm. Second, we compare two missing pattern detection algorithm. They are identical except the clustering part. We measure the running time in second to show that missing pattern detection algorithm hold the same running time whether using DBSCAN or LINEAR-CLUSTER. The DBSCAN algorithm in both experiment is from *fpc* library [9] for *R* language, which has $O(n^2)$ running time. And all experiments were done using *R* in order to reflect real system environment at HAII.

A. Experiment on DBSCAN and Linear-Cluster

In this experiment, we benchmark the running time of DBSCAN and LINEAR-CLUSTER directly. Dataset is generate by using continuous function to reflect the nature of water-level data. Then, we randomly shift some data points in order to generate outliers. Specifically, we use $y = sin(0.005x)$. Then we randomly pick 10 percent of data point and add them with random number generated using normal distribution (sd = 1). Example of generated data is shown in Fig. 7. We test both algorithm from 10,000 data points to 30,000 data points. A result is shown in Table I and Fig. 8. From the result, the LINEAR-CLUSTER is significantly faster than DBSCAN.

B. Experiment on Missing Pattern

In this experiment, we use real data from a telemetry system. The data selected is from the telemetry station named CHI005, between October 2011 and September 2013. We feed this data into two missing pattern detection algorithm using different clustering algorithm. The algorithm is executed 10 times for each data length and mean execution time is measured. A

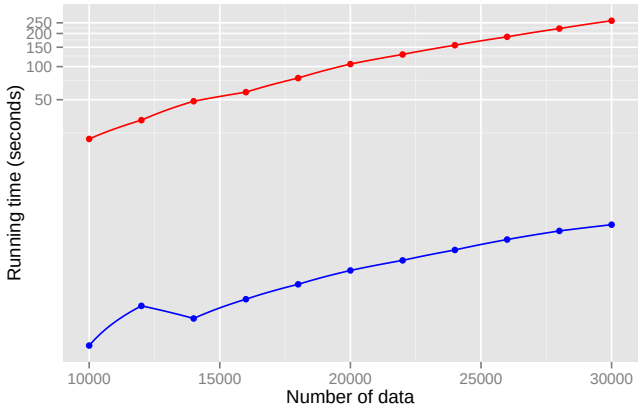

Fig. 8. Running time between LINEAR-CLUSTER (blue) and DBSCAN (red). Y-axis is plotted in binary log scale.

TABLE I
RUNNING TIME IN SECOND OF DBSCAN AND LINEAR-CLUSTER

Number of data	**DBSCAN** (s)	**Linear-Cluster** (s)
10,000	21.978	0.291
12,000	32.593	0.668
14,000	48.419	0.513
16,000	58.566	0.768
18,000	78.615	1.049
20,000	105.393	1.402
22,000	128.891	1.731
24,000	156.447	2.152
26,000	186.461	2.678
28,000	221.635	3.206
30,000	261.015	3.654

result is shown in Table II and Fig. 9. From the result, the LINEAR-CLUSTER holds the same running time as DBSCAN with insignificant difference.

V. CONCLUSION

The paper propose to improve the clustering algorithm used in both outliers detection algorithm and missing pattern

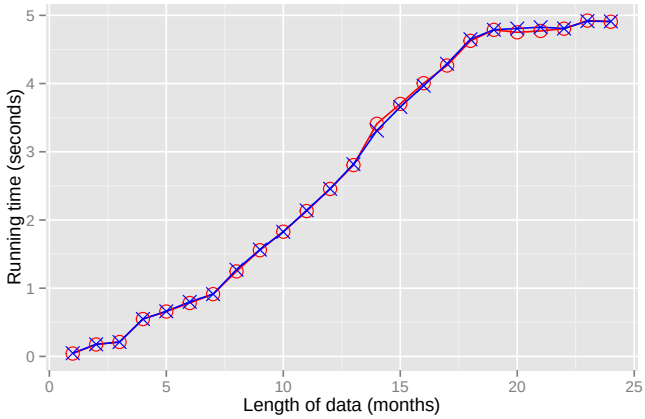

Fig. 9. Running time of missing pattern detection algorithm using DBSCAN (blue, cross sign) and using LINEAR-CLUSTER (red, circle sign).

TABLE II
MEAN RUNNING TIME OF MISSING PATTERN DETECTION ALGORITHM USING DBSCAN AND LINEAR-CLUSTER

Length of data (month)	**DBSCAN** (s)	**Linear-Cluster** (s)
1	0.0420	0.0472
2	0.1751	0.1773
3	0.2111	0.2107
4	0.5476	0.5480
5	0.6581	0.6615
6	0.7830	0.7996
7	0.9129	0.9133
8	1.2466	1.2711
9	1.5567	1.5652
10	1.8283	1.8262
11	2.1312	2.1410
12	2.4553	2.4581
13	2.8046	2.8177
14	3.4103	3.3106
15	3.7018	3.6573
16	4.0058	3.9693
17	4.2645	4.2896
18	4.6262	4.6498
19	4.7870	4.7891
20	4.7509	4.8072
21	4.7713	4.8262
22	4.8047	4.8076
23	4.9229	4.9153
24	4.9063	4.9114

detection algorithm in our previous work. The linear clustering algorithm is created to replace DBSCAN. Outliers detection algorithm now have better running time at $O(n)$. Also, missing pattern detection algorithm holds the same running time at $O(n \log n)$. Moreover, we conduct experiments to compare the running time of both algorithms. The result conform to the theoretical running time.

REFERENCES

[1] B. Post, "The world bank supports thailand's post-floods recovery effort," http://www.worldbank.org/en/news/feature/2011/12/13/world-bank-supports-thailands-post-floods-recovery-effort, dec 2011, accessed: 2015-04-01.

[2] Hydro and Agro Informatics Institute, "Background," http://www.haii.or.th/haiiweb/index.php?option=com_content&task=view&id=94&Itemid=95&lang=en, accessed: 2015-04-01.

[3] ——, "Mobile telemetry system," http://www.thaiwater.net/web/index.php/aboutusen/524-telemeterringenen.html, accessed: 2015-04-01.

[4] P. Markpeng, P. Wongnimmarn, N. Champreeda, P. Vateekul, and K. Sarinnapakorn, "Controlling quality of water-level data in thailand," in *Intelligent Information and Database Systems*. Springer, 2014, pp. 503–512.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, *The R*-tree: an efficient and robust access method for points and rectangles*. ACM, 1990, vol. 19, no. 2.

[7] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2011, ch. 4.3.

[9] C. Hennig, *Flexible procedures for clustering*, 2nd ed., jan 2014.