Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 3 (2011) 26-38

Analysis of an Event Forecasting Method for Wireless Sensor Networks

András KALMÁR, Gergely ÖLLÖS, Rolland VIDA

High Speed Networks Laboratory (HSNLab),
Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
e-mail: {kalmar; ollos; vida}@tmit.bme.hu

Manuscript received November 10, 2011; revised December 15, 2011.

Abstract: In this paper we introduce an event forecasting method for wireless sensor networks (WSNs), and its testing results in real world circumstances. The algorithm recognizes and differentiates the event sequences that turn up inside the sensor field, and then uses these recognized event sequences for event forecasting. The method uses the Fuzzy set theory and clustering methods. The events are represented with three different parameters (measurement data, sensor ID, and timestamp). According to the model of the algorithm, each sensor node periodically samples a predefined environmental parameter, and if the value of the measurement data is higher than a predefined threshold, the node stores this data as an event. A series of events is stored in the so called Time-Space fuzzy Signature (TSS). A TSS is a set of events, which are detected on a local node, or on its neighbors in its communication range. The algorithm performs hierarchical clustering on the TSSs to determine, which of them can represent the same event sequence, and as a result the same phenomenon. Then, on the results of the hierarchical clustering we perform a K-mean clustering, in order to filter out the noise events. The event forecasting feature of the WSN can be useful for target tracking or sleep scheduling protocols, among others. In this article, first we shortly introduce the theoretical background and definitions of the algorithm; then, we demonstrate the working.

Keywords: Wireless Sensor Network, event forecasting, fuzzy theory, clustering.

1. Introduction

A Wireless Sensor Network (WSN) consists of a large number of distributed nodes that organize themselves into a multi-hop wireless network. Each node has one or more sensors, embedded processors and a low-power radio. Typically, these nodes coordinate to perform a common task. The WSNs greatly extend our ability to monitor and control the physical environment from remote locations; furthermore, they can greatly improve the accuracy of information obtained via collaboration between sensor nodes. An interesting and useful feature of these networks is called event forecasting.

In [1], we suggested an event forecasting algorithm for WSNs, which builds up on a fuzzy framework. According to this method the sensor nodes try to forecast specific events from the changes of the environmental parameters, and from formerly registered measurements. This feature of the WSNs can be very useful in practice. The method is fully distributed and robust, and it does not require hard time synchronization or localization. In this article we would like to introduce an algorithm which is based on the same fuzzy framework, but tries to recognize and differentiate the event sequences that occur inside the area covered by the sensor network, and extract from them the “pure sequence(s)”. (*A pure sequence is an event sequence that occurs inside the area monitored by the sensor network just because of a phenomenon and does not contain noise events.*) These pure sequences can be used then for event forecasting. The efficiency of the proposed method was tested in real circumstances as well, using a few Crossbow MicaZ sensor nodes that were placed next to different kinds of crossroads; the event-sequences recognized by the sensors model well the different trajectories of the passing cars.

In the next section we introduce the theoretical background, how the events are represented on the sensor nodes, and how the nodes store and sort the event sequences that appear in the network.

2. Definitions

In the following we provide a few definitions that are necessary to understand the context in which the forecasting model was developed and the measurements were done. These definitions and notations will be further detailed, and their usage will be explained in the following sections.

In [1] we defined a fuzzy set of events as follows:

$$E = \{(f, \mu_E(.), ID_f, t_f) \mid f \in F\} \tag{1}$$

where F is a certain feature space of the taken measurement. In Fuzzy terminology F is the universe of the features (*f*), where the events ($e \in E$) are defined. The parameter $\mu_E(.)$ is called fuzzy membership function. This function assigns to every f a number between 0 and 1, depending on the degree at which *f* belongs to the fuzzy set of events. ID_f means the *ID* of the sensor node where the event appeared, and t_f is the detection time of the event.

In the case when a node has more than one event to be managed, we store these events in the so called Time-Space fuzzy Signature (TSS)

$$\boldsymbol{TSS}_{\boldsymbol{ID,i}} = \{e_{trg}, e_1, e_2, ..., e_n \mid e \in E\} \tag{2}$$

This is a set of events that occurred before the ***i***-th target event (e_{trg}) being detected by the local sensor node ($\boldsymbol{ID}$). The $e_1, e_2, ..., e_n$ events are either events detected by neighboring sensors, which alerted all the other nodes in their vicinity, or they are events detected by the local node itself, at a previous moment in time. These events are sorted in the TSS in descending order, by their time of occurrence (t_f).

3. Problem formulation

According to the model, each sensor node periodically samples a predefined environmental parameter, and in the case when the value of the membership function assigned to this environmental parameter is higher than a threshold limit, the node stores this data as an event. The shape of the membership function can be defined arbitrarily, according to a specific interval in the input space, based on what the user regards as an event. When a node detects an event, it sends a “limited broadcast” message to its neighboring nodes. Every sensor in radio range receives this message. All the sensors have a TSS database, which is filled up with their own events and the events detected by the neighboring nodes. A phenomenon passing through the monitored area creates event sequences (called *global event sequences*).

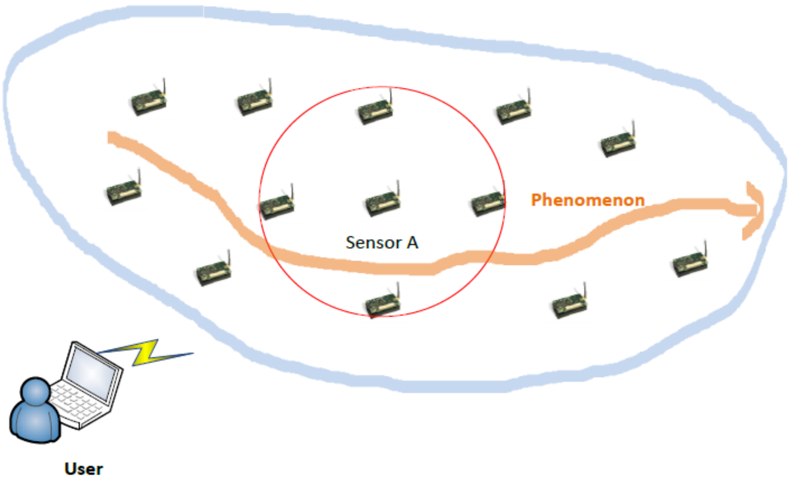

Figure 1: A local event sequence.

From this global event sequence a simple node senses only those events, which appeared in its communication range (called *local sequence*); in *Fig. 1* sensor node A sees only those events, which are logged in his communication range (red circle). There can be multiple phenomena in the sensor field at the same time, so the global sequences, and as a consequence the local sequences as well, can be overlapped with each other. In that case, the nodes can filter out from this mixed event set the pure local sequences, and then these pure local sequences can be used for event forecasting.

4. The TSS distance

As mentioned before, a phenomenon passing through the area monitored by the sensor network creates event sequences. Our aim is to estimate the number of the "pure sequences" that are mixed due to the overlapped sequences, i.e., how many different phenomena affected the sensor nodes. We have to find the similar event sequences, and assign them to the same cluster group. Then, we should extract from each cluster group the "pure event sequence", i.e., what phenomenon that group represents. To gain the ability to create clusters from the mixed event sequences, we must define a distance function between the different TSSs. But before doing so, in order to be able to compare the TSSs, that were created at different moments in time, we have to normalize the TSSs by time.

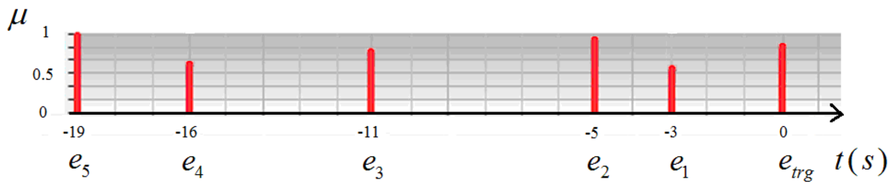

Figure 2: A normalized TSS.

Fig. 2 illustrates a TSS after normalization. As shown in the picture, normalization means that we shift the time parameter of the events so that the target event is at time 0, and all the other events, which occurred earlier in time than the target event, have negative time parameters. With this normalization we can transform all TSSs to a standard form.

The TSS distance function compares two normalized TSSs and assigns a number between 0 and 1 to them, in order to describe the similarity of the two TSSs. This is carried out in two steps. In the first step we try to order into pairs the events of the TSSs, while in the next step we try to quantify the differences between the pairs that we have found. One of the most important aspects in looking for event pairs is that we only search pairs in events which have the same sensor ID.

Fig. 3 illustrates a case, when two TSSs contain only events that are related to two different sensor IDs (sensorID = 1 and 2). In the figure we can see a possible pairing among events with sensorID = 1, but we also see that another pairing is possible too. Finding the best possible pairing is important, because we can filter out the noise events, as these events won't have pairs in the pairing. We can draw one important conclusion, that the time difference of the found pairs should be minimal.

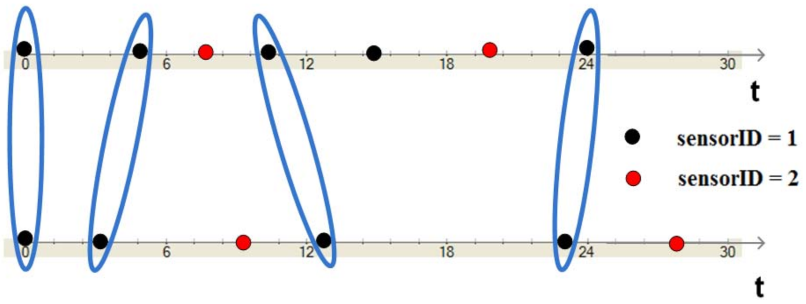

Figure 3: A possible event pairing.

Fig. 4 illustrates another case, where we can see that taking into account only the minimization of the time differences, as mentioned above, might not lead to an optimal solution. The reason for it is that the event sequences of the two TSSs might be shifted a bit in time. This pair choosing method will thus not pair together those point pairs which really belong together.

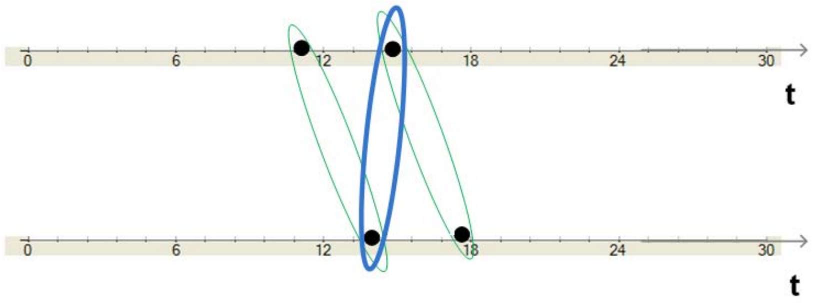

Figure 4: An example when the above greedy pair choosing algorithm does not get an optimal solution.

Thus, we have to redefine the conclusion above so that in the pairing the sum of the time differences should be minimal. This problem is equivalent with the assignment problem in graph theory, which consists in finding a maximum weight matching in a weighted bipartite graph. This assignment problem can be solved in polynomial time, for instance with the Hungarian method [3].

The weights of the edges are

$$w = \frac{1 - |\Delta\mu|}{\Delta t^2} \tag{3}$$

where, w is the weight of the edge, Δt is the time difference between two events, while $\Delta\mu$ is the μ difference between the two events. We can see that the weight of an edge between two events depends on the time and the membership function differences between the events. The weights should have a maximum and a minimum value. On one hand with the maximum value we can normalize the sum of the weights; on the other hand with the minimum value we can avoid the case when we should divide with zero according to the formula above.

At this point we have a weighted bipartite graph and we must search the maximum weight matching in the sub-graphs assigned to the different sensor IDs. Then, the distance between two TSSs is calculated as follows:

$$d = 1 - \frac{\sum w}{TSS_{length,\min} * w_{\max}} \tag{4}$$

where *w* denotes the weights of the different edges, while TSS $_{\text{length, min}}$ is the number of elements of the shorter TSS.

5. The event forecasting method

According to the model, each sensor node periodically samples a predefined environmental parameter, and if this sampled value is higher than the threshold limit, the node stores this data as an event. Every sensor has a TSS database, the purpose of the algorithm is thus to create cluster groups from the stored TSSs, and then to extract from these groups the pure event sequences. We used a hierarchical clustering solution, and the results are illustrated with a dendrogram, as shown in *Fig. 5*.

The root node of the dendrogram represents the whole TSS database, and each leaf node is regarded as a TSS. The intermediate nodes thus describe the extent to which the objects are similar to each other; while the height of the dendrogram expresses the distance between each pair of TSSs or clusters, or a TSS and a cluster.

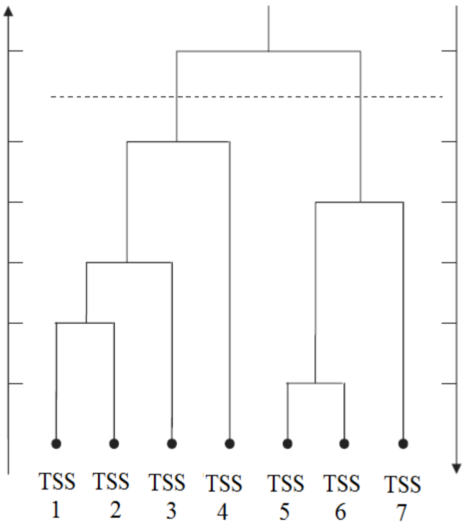

Figure 5: A dendrogram representing the result of the hierarchical clustering.

After the formation of the hierarchical cluster set from the TSS database, in order to attain the desired subsets the dendrogram should be cut at the proper levels. These subsets contain such TSSs, which represent the same event sequence mixed with noise events. After these subsets are available, the algorithm tries to extract the pure event sequences from them with the so called k-mean clustering.

K-mean clustering is one of the hard partitioning methods. It searches an optimal partition of the data by minimizing the sum-of-squared-error criterion (4) in an iterative optimization procedure [2].

$$J_s(\Gamma, M) = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} \| x_j - m_i \|^2 \tag{4}$$

Let $\Gamma=\{\gamma\}_{ij}$ be the partition matrix, defined as follows:

$$\gamma_{i,j} = \begin{cases} 1, \text{ } if \ x_j \in C_i \\ 0, \text{ } otherwise \end{cases} \quad \text{and} \quad \sum_{i=1}^{K} \gamma_{ij} = 1, \forall j$$

Let $M = [m_1, ... m_k]$ be the cluster centroid matrix, where $m_i = \frac{1}{N_i} \sum_{j=1}^{N} \gamma_{ij} x_j$ is the sample mean for the i^{th} cluster with N_i objects.

The steps of the *k*-mean clustering algorithm are then the following:

1. Initialize a K - partition randomly, or based on some prior knowledge. Calculate the cluster centroid matrix $M = [m_1, ... m_k]$
2. Assign each object in the data set to the nearest cluster C_a , i.e.,

$$x_j \in C_a \text{, if } \| x_j - m_a \| < \| x_j - m_b \|$$

$$j = 1, ..., N, \ a \neq b, \ a = 1, ..., K \text{ and } b = 1, ..., K$$

3. Recalculate the cluster centroid matrix based on the current partition,

$$m_a = \frac{1}{N_a} \sum_{x_j \in C_a} x_j$$

4. Repeat steps 2 and 3 until there is no change in any cluster.

In *Fig. 6* we can see the previously detailed steps for a two-dimensional case. The algorithm randomly assigns two cluster centroids to the input points. Each data point is assigned to a cluster centroid according to the predefined distance function. In the next step the centroids are recomputed. The clustering method executes these steps repeatedly, until there is no change in the centroid matrix. Returning to the forecast algorithm, as a result of cutting the dendrogram at the proper levels, the desired TSS subsets turn up. These subsets will contain TSSs which represent probably the same event sequence mixed with noise events. In that case, we divide the different events assigned to different sensor IDs in the subsets, and we group the three-dimensional (sensor

ID, timestamp, μ) event data structures into two-dimensional planes. These planes represent events which possess the same sensor ID. On those planes we use the formerly detailed k-mean algorithm, as follows: we start with one cluster group, and in each iteration we increase the number of the cluster groups. During an iteration we analyze two conditions: the variances of the groups, and the number of events in each group. If the variance of a group is less than a predefined parameter, it means that at this place in that plane (time stamp, μ) the events are placed densely, and these events can represent a pure event sequence. To verify this, we have to compare the number of events in this group with the number of the TSSs in the subset (dendrogram sub-tree).

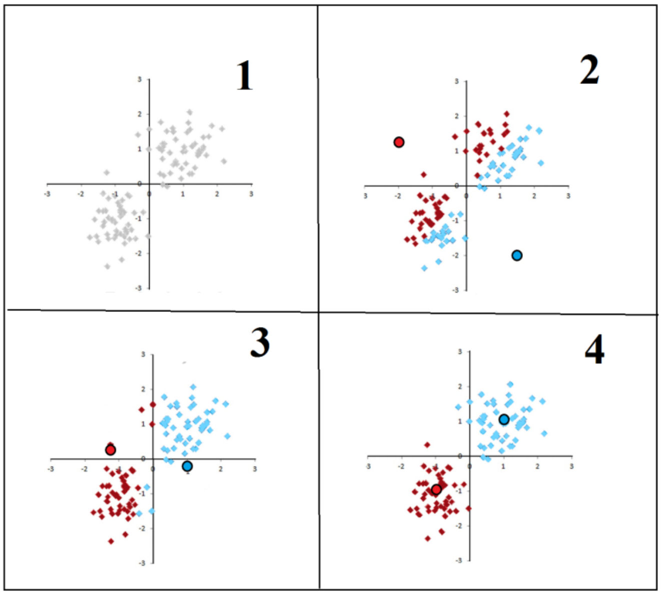

Figure 6: Steps of the K-mean clustering algorithm.

If their ratio is above a second predefined threshold, it means that we have found a pure event sequence, so we store the centroid (time stamp, μ) of the group with the sensor ID assigned to that plane. After that we continue the process on the next plane.

6. Performance analysis

The previously introduced event forecasting algorithm was tested in real circumstances as follows: we implemented a WSN with user interface from

Crossbow MicaZ sensor nodes, with the purpose to measure sound intensity in different kinds of crossroads. After the measurement, the measured data from each sensor node was uploaded to a WSN simulator that was running on a PC. The reasons why the testing of the algorithm was carried out in a simulator and not on the sensor nodes, are the relatively small program memory available on the nodes, the cumbersome nature of the debugging process on the real sensors, and the more clear and transparent supervision of the processes in a simulator. The main purpose of this experiment was to determine whether the event-sequences recognized by the sensors are modeling well the different trajectories of the passing cars.

One directional, straight road

Probably the simplest case is when the nodes are placed along a one directional, straight road. It is simple, because there is only one event sequence to be recognized. The potential difficulties in this case are the following. The different speeds of the cars along the road result in time shifts in the searched event sequences. The different sound intensities of the various cars cause offsets in the membership values of the events. In addition, the acceleration changes of the vehicles cause both of these problems. The setting of the nodes along the road can be seen in *Fig. 7*.

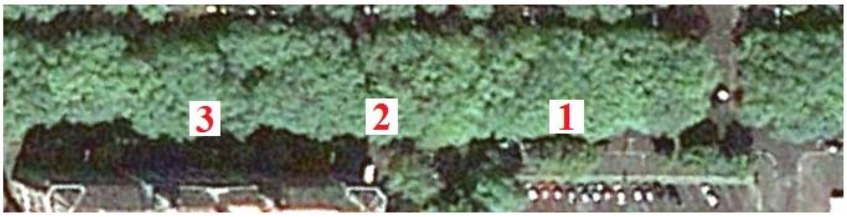

Figure 7: Setting of the nodes along the one directional straight road.

The red numbers mark the sensor IDs of each sensor. The distance between the nodes was approximately 20 meters. As we can see in the figure, the searched event sequence was the (3-2-1). The algorithm recognized this event sequence clearly, and in addition it recognized it in multiple forms, in the sense that in each form the order of the sensor IDs was the same, but the time differences between the events were different. On the whole it can be said that these event sequences (assigned to vehicles with different speeds) were differentiated.

Two directional, straight road

The second measurement was made along a two directional straight road. It was similar to the one directional case in the sense that all of the nodes were planted on the same side of the road in a line, the distance between the nodes being 20 meters. The setting of the nodes can be seen in *Fig. 8.*

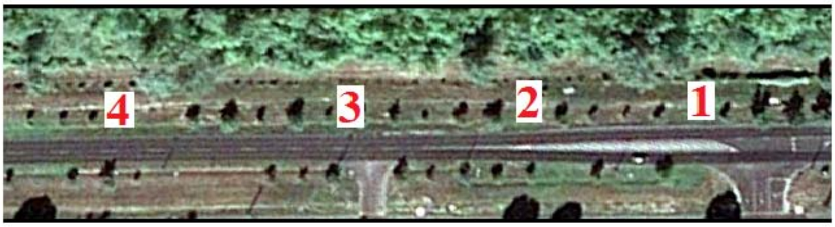

Figure 8: Setting of the nodes along the two directional straight road.

The difficulties to recognize the event sequences were the same as before (speed and acceleration changes), but in this case there were two event sequences to identify, caused by the cars moving on the two sides of the road. These event sequences overlapped with each other in most of the cases. The two event sequences were the (1-2-3-4) and the (4-3-2-1).

From the (1-2-3-4) event sequence the 4th sensor found the (2-3-4), the 3rd node the (2-3) and the 2nd sensor the (1-2) event sequences. There could be several explanations, why the nodes found only event "sub-sequences" from the complete sequence. The overlapping event sequences, the not ideal TSS length, or the noisy environment could all be the reasons for this.

From the (4-3-2-1) event sequence the nodes typically stored only their own sensed event, and that of the preceding sensor node. The recognized sequences were the following: (4-3), (3-2), and (2-1). Considering the fact that the nodes could sense the events of the distant lane less efficiently, and the events of the closer lane could fade them, it is most likely that the above mentioned possible error sources could have a greater impact on the results in this case.

Crossroad

After the straight road measurements, we analyzed a crossroad, which had an average traffic intensity. This case was the most difficult so far, as there were several event sequences to be recognized. If a vehicle tried to leave the horizontal main road for the vertical low priority road, it might have taken several seconds to carry out its task. This time interval depended on the actual traffic conditions. This means that the time interval of the same event sequences could change. The topology of the measuring nodes can be seen in *Fig. 9.*

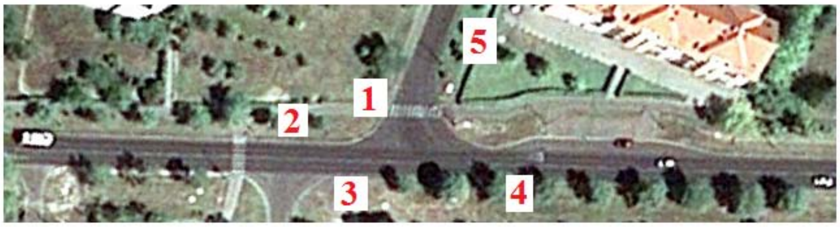

Figure 9: The placement of the nodes in the crossroad.

The recognized event sequences were the following: the 2nd node found the (4-3-2) sequence, and was able to mark the vehicles traveling horizontally from right to left on the main road. The 3rd sensor identified the (2-3), the 4th node found the (3-4), and the 5th node found the (4-5) sequences. The (2-3) and the (3-4) event sequences could mark the cars moving horizontally from left to right on the main road. The (4-5) sequence possibly marks vehicles, which turn down from the main road to the lower priority road. The 1st sensor node registered very few events and it didn't found any sequence as a result. The detected event sequences characterize the trajectories of the passing cars quite well.

To summarize the results it can be said that the recognized event sequences contain only two or three events. The density of the events and the small number of the nodes could cause this. In the case if the sensor nodes send these recognized local sequences to a base station, and this fits them together, than the desired global event sequences turn up.

6. Conclusion

In this paper we introduced an event forecasting method for wireless sensor networks and presented its testing results in real circumstances. With Crossbow MicaZ sensor nodes we measured the sound intensity of the vehicles next to various types of roads and in a crossroad. Our purpose was to ensure that the event-sequences recognized by the sensors model well the different trajectories of the passing cars. The experiments showed that in most of the cases the recognized event sequences contained only two or three events. This was probably caused by the small number of the used nodes and the density of the events. If a base station can communicate with all the nodes, then it has the ability to fit together these recognized event sequences and identify the global event sequences. These global sequences mark well the different phenomena appearing in the field of the WSN.

References

[1] Öllös, G., Vida, R., "Adaptive Event Forecasting in Wireless Sensor Networks", in *IEEE Vehicular Technology Conference* (*IEEE VTC2011-Spring*), Budapest, Hungary, May 15-18, 2011.

[2] Xu, R., Wunsch, D. C., "Clustering", John Wiley and Sons, New Jersey, 2009.

[3] Kuhn, H. W., "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly*, Vol. 2, Issue 1-2, March 1955, pp. 83–97.

[4] Jang, J. -S. R., Sun, C. -T., Mizutani, E., "Neuro-Fuzzy and Soft Computing", Prentice-Hall Inc., 1997.

[5] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., Cayirci, E., "Wireless Sensor Networks: a Survey", *Computer Networks*, Vol. 38, pp. 393-422., 2002.

[6] L. A. Zadeh "Fuzzy sets", *Information and Control*, Volume 8, Issue 3, June 1965, pp. 338-353.