Fault handling in large water networks with online dictionary learning

Paul Irofti [a,*,1], Florin Stoican [b], Vicenç Puig [c]

[a] Department of Computer Science and the Research Institute of the University of Bucharest (ICUB), Romania
[b] Department of Automatic Control and Computers, University Politehnica of Bucharest, Romania
[c] Universitat Politècnica de Catalunya, Institut de Robòtica i Informàtica Industrial (CSIC, UPC), Barcelona, Spain

ARTICLE INFO

Article history:
Received 17 March 2020
Received in revised form 18 July 2020
Accepted 23 August 2020
Available online 7 September 2020

Keywords:
Fault detection and isolation
Sensor placement
Online dictionary learning
Classification
Water networks

ABSTRACT

Fault detection and isolation in water distribution networks is an active topic due to the nonlinearities of flow propagation and recent increases in data availability due to sensor deployment. Here, we propose an efficient two-step data driven alternative: first, we perform sensor placement taking the network topology into account; second, we use incoming sensor data to build a network model through online dictionary learning. Online learning is fast and allows tackling large networks as it processes small batches of signals at a time. This brings the benefit of continuous integration of new data into the existing network model, either in the beginning for training or in production when new data samples are gathered. The proposed algorithms show good performance in our simulations on both small and large-scale networks.

1. Introduction

Water distribution networks (along electricity, transport and communication ones) are a critical infrastructure component. Thus, modeling and observability issues are of paramount importance and have to be handled through increases in connectivity, automation and smart metering.

In particular, pipe leakages (assimilated hereinafter with fault events) have to be detected and isolated as soon and as precisely as possible. While apparently straightforward (the leakage leads to a measurable loss of pressure), several issues conspire in increasing the scheme's difficulty:

(i) disparity between the number of nodes (hundreds /thousands in a large-scale network) and of sensors (expensive/hard to install, hence at most tens) [1–3];
(ii) water network dynamics are strongly nonlinear and demand uncertainties are significant [4,5].

Hence, sensors have to be placed to provide network-wide relevant information while pressure and flow information is obtained either through emulation [6] or experimentally [7]. Such data driven analysis naturally leads to heuristic implementations which come with specific caveats:

(i) heuristic methods use the data agnostically and ignore information about network structure/particularities;
(ii) network size may lead to implementations which clog the resources or are bogged by numerical artifacts.

In light of the previous remarks, it is clear that the main issues are sensor placement and subsequent fault detection and isolation (FDI) mechanism. For the former, we propose a novel Gram–Schmidt graph-aware procedure and for the later we consider a dictionary learning (DL) approach.

Specifically, we assign the faults affecting a given node to a class and train a dictionary such that its atoms discriminate between these classes. The subsequent classification of a signal in terms of the dictionary's atoms serves as proxy for FDI. The active atoms for a certain class are seen as a fault signature which unambiguously asserts FDI (if the signature is unique w.r.t. the other possible faults).

DL [8] is an active research topic in the signal processing community providing machine learning algorithms that build linear models based on the given (obtained from processes with nonlinear dynamics) input data. Its applications to classification tasks [9] in general and online classification [10] in particular provide fast and memory efficient implementations well suited for IoT devices and for online production usage. Furthermore, DL methods are known to be robust to various types of noises and perturbations [11], a key property for our application.

Our previous work [12,13] has shown encouraging results when adapting DL classification for FDI in water networks. In this paper, we propose new methods that tackle large distribution networks and lift data dimensionality limitations by employing online DL strategies (which process data in small batches which translate into smaller computation complexities). This leads to a small decrease of the FDI performance as compared to a method

* Corresponding author.
E-mail addresses: paul@irofti.net (P. Irofti), florin.stoican@acse.pub.ro (F. Stoican), vicenc.puig@upc.edu (V. Puig).

[1] This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS - UEFISCDI, project number PN-III-P1-1.1-PD-2019-0825, within PNCDI III.

handling the whole data set at once, however this is quickly attenuated as more data is processed.

In simulations, we consider both the proof-of-concept benchmark "Hanoi network" and a generic large-scale network [14]. Furthermore, we use multiple demand profiles, fault magnitudes and discuss different sensor placement strategies and success criteria.

2. Preliminaries

A passive water network (i.e., without active elements like pumps) consists of one or more tank nodes (whose heads[2] remain constant) which feed a collection of junction nodes through a network of interconnected pipes. From a modeling perspective, the question is what are the flows through the pipes, what are the heads through the junction nodes and how do these variables depend on user demand (outflows from some or all of the junction nodes) and unexpected events (in our case: pipe leakages).

2.1. Steady-state behavior

The dynamics of the network are usually ignored. This is a reasonable assumption as long as demand variation is slow and unexpected events (e.g., leakages) are rare. In other words, any transient-inducing event is sufficiently rare and the transients themselves are sufficiently fast such that it is a fair approximation to consider the system at equilibrium [5]. Since water is incompressible the relevant physical laws which apply are those of mass and energy conservation. First, the inflows and outflows passing throughout a junction node have to balance:

$$\sum_{j=1}^{n} \mathbf{B}_{ij}\mathbf{q}_j = \mathbf{c}_i \tag{1}$$

where $\mathbf{q}_j$ is the flow through pipe j, $\mathbf{c}_i$ is the consumption of node i and $\mathbf{B}$ is the adjacency matrix of the network, i.e., $\mathbf{B}_{ij}$ takes one of the following values:

$$\mathbf{B}_{ij} = \begin{cases} 1, & \text{if pipe } j \text{ enters node } i; \\ 0, & \text{if pipe } j \text{ is not connected to node } i; \\ -1, & \text{if pipe } j \text{ leaves node } i. \end{cases} \tag{2}$$

Next, the empiric Hazen–Williams formula [15] gives the head flow variation between nodes i, j linked through a pipe with index ℓ (we assume that the pipe of index ℓ links the i, jth nodes):

$$\mathbf{h}_i - \mathbf{h}_j = \frac{10.67 \cdot L_\ell}{C_\ell^{1.852} \cdot D_\ell^{4.87}} \cdot \mathbf{q}_\ell \cdot |\mathbf{q}_\ell|^{0.852} \tag{3}$$

where L_ℓ is the length in [m], D_ℓ is the diameter in [m] and C_ℓ is the adimensional pipe roughness coefficient; the flow $\mathbf{q}_\ell$ is measured in [m^3/s].

Using (3) we express the flow in terms of the associated head flow $\mathbf{h}_i - \mathbf{h}_j$:

$$\mathbf{q}_\ell = G_{ij}^{0.54}(\mathbf{h}_i - \mathbf{h}_j)|\mathbf{h}_i - \mathbf{h}_j|^{-0.46} \tag{4}$$

where G_{ij} is the pipe conductivity, defined as

$$G_{ij} = \frac{1}{R_{ij}} = \frac{C_\ell^{1.852} \cdot D_\ell^{4.87}}{10.67 \cdot L_\ell}. \tag{5}$$

Noting that the ℓ-th line of the column vector $-\mathbf{B}^\top\mathbf{h}$ returns the difference $\mathbf{h}_i - \mathbf{h}_j$ and combining (1) with (4) leads to the nonlinear steady-state relations:

$$\mathbf{BG}\left[\left(-\mathbf{B}^\top\mathbf{h} + \mathbf{B}_f^\top\mathbf{h}_f\right) \times \left|-\mathbf{B}^\top\mathbf{h} + \mathbf{B}_f^\top\mathbf{h}_f\right|^{-0.46}\right] = \mathbf{c}, \tag{6}$$

where $\mathbf{G} = \text{diag}\,(G_\ell)$ and '$\times$' denotes the elementwise multiplication of two vectors (i.e., the ith element of $x \times y$ is $[x \times y]_i = x_i y_i$). Note the addition of term $\mathbf{B}_f^\top\mathbf{h}_f$ which describes the influence of fixed-head nodes (the tanks which feed the network). For further use, we denote with N the number of junction nodes.

2.2. Node consumption

Assuming that all the parameters of (6) are known (gathered in the left side of the equation), there still remains the node consumption $\mathbf{c}$ as a source of uncertainty. Historically user demand data has been acquired sparsely or not at all. Most common approaches are to consider a consumption profile (usually with a week-long period) and scale it wrt the total consumption in the network:

$$\mathbf{c}_i(t) = \frac{\bar{\mathbf{c}}_i}{\sum_j \bar{\mathbf{c}}_j} \cdot p(t) \cdot \mathbf{q}_{in}(t) + \boldsymbol{\eta}_i(t), \tag{7}$$

where $p(t)$ and $\mathbf{q}_{in}(t)$ are the consumption profile and, respectively, the total water fed to the network at time instant t; $\bar{\mathbf{c}}_i$ denotes the base demand for the ith node and $\boldsymbol{\eta}_i(t)$ covers 'nominal' (those occurring under healthy functioning) uncertainties affecting the ith node (without being exhaustive: normal user variations, seasonal and holiday variations, small leakages).

The issue of interest is how to detect and isolate a pipe leakage. We note that a pipe leakage means a loss of flow and thus a loss of head in the network's nodes. We then interpret pipe leakage as an additive term in the node consumption value[3]:

$$\mathbf{c}_i(t) = \frac{\bar{\mathbf{c}}_i}{\sum_j \bar{\mathbf{c}}_j} \cdot p(t) \cdot \mathbf{q}_{in}(t) + \boldsymbol{\eta}_i + \mathbf{f}_i. \tag{8}$$

For further use, we consider that the profile $p(t)$ can take values from a collection of P profiles $\{p_1(t), \ldots, p_P(t)\}$.

Remark 1. This means that the active profile in (7)–(8) may be unknown at measuring time. This additional uncertainty may hide water losses due to pipe leakages. The preferred solution is in practice to measure total water demand $\mathbf{q}_{in}(t)$ at times when user demand is minimal (middle of the night). At this time, deviations due to leakages represent a larger percentage from the total consumption (wrt the uncertainty due to the profile). Thus, a change from the expected value may signify that leakages are present (in FDI parlance, a fault is detected). ♦

2.3. Leakage isolation and residual generation

The issue of leakage isolation still remains. To assess the leakage events we have to compare the "healthy" (nominal) behavior, as given in (7), with the measured (and possibly faulty, as given in (8)) behavior of the network's nodes. This is done through a *residual signal* which is [16]: (i) constructed from known quantities; (ii) sensitive to a fault occurrence[4]; and (iii) robust, in as much as is possible, to normal variations.

For further use we make a couple of assumptions.

Assumption 1. We consider that there are no multiple fault occurrences in the network (i.e., the network is either under nominal functioning or with a single node under fault). ♦

[2] In the water network parlance, "head" denotes the height of the column of water in a node wrt a common ground level.

[3] Hereafter when we speak about isolating a leakage we refer to identifying the node directly affected by the pipe leakage. The actual leakage isolation means checking the pipes which enter into the node.

[4] Hereinafter, to keep with the FDI context we denote a 'leakage event' as a 'fault occurrence'.

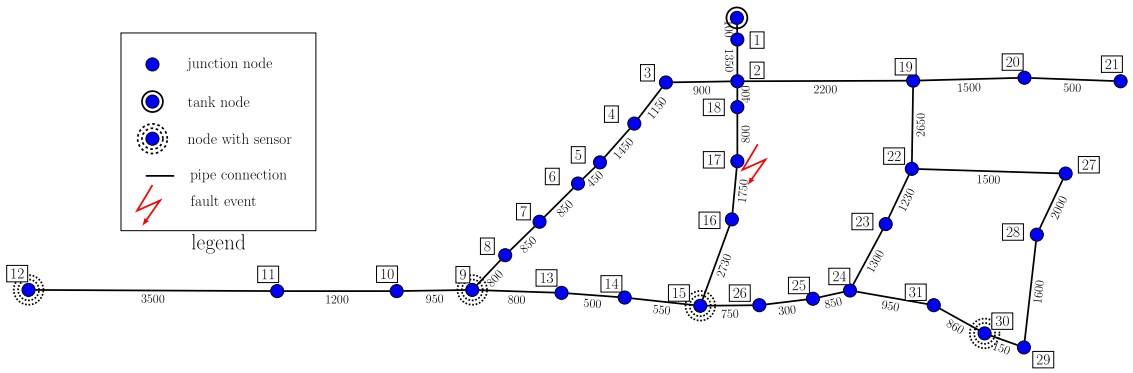


Fig. 1. Hanoi water network.

Assumption 2. Without loss of generality we assume that the fault magnitude values are the same for each node and are taken from a finite collection (M possible values from $\{m_1, \dots m_M\}$). ♦

For further use we consider the nodes' head as a proxy for fault occurrences and use its nominal ($\bar{\mathbf{h}}$) and measured values ($\hat{\mathbf{h}}$) to construct the residual signal. The following aspects are relevant:

- as per Remark 1, we consider an interval K in which the head values remain relatively constant and average over it to obtain the "steady-state" nominal/measured head values:

$$\bar{\mathbf{h}} = \frac{1}{|K|}\sum_{k\in K}\bar{\mathbf{h}}[k], \quad \hat{\mathbf{h}} = \frac{1}{|K|}\sum_{k\in K}\hat{\mathbf{h}}[k]. \tag{9}$$

- the residual may be defined in absolute or relative form, i.e.:

$$\mathbf{r}^A = \hat{\mathbf{h}} - \bar{\mathbf{h}}, \quad \mathbf{r}^R = \frac{\hat{\mathbf{h}} - \bar{\mathbf{h}}}{\bar{\mathbf{h}}}. \tag{10}$$

  Whenever the residuals' type (absolute or relative) are not relevant we ignore the superscripts 'A, R'.
- assuming that the ith node is under fault with magnitude m_j and that the network functions under profile $\mathbf{p}_k$, we note[5] the head values with $\hat{\mathbf{h}}_{ij}^k$ and corresponding residual with $\mathbf{r}_{ij}^k$.

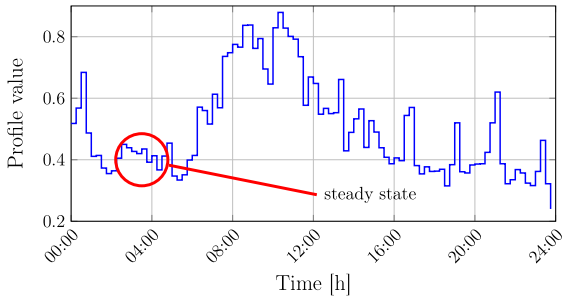
For further use we gather all the residual vectors $\mathbf{r}_{ij}^k$ into the *residual matrix*[6] $\mathbf{R} \in \mathbb{R}^{N\times D}$:

$$\mathbf{R} = \Big[\underbrace{\mathbf{r}_{11}^1\, \mathbf{r}_{12}^1 \dots \mathbf{r}_{1M}^1 \dots \mathbf{r}_{11}^P\, \mathbf{r}_{12}^P \dots \mathbf{r}_{1M}^P}_{f_1} \dots$$
$$\underbrace{\mathbf{r}_{N1}^1\, \mathbf{r}_{N2}^1 \dots \mathbf{r}_{NM}^1 \dots \mathbf{r}_{N1}^P\, \mathbf{r}_{N2}^P \dots \mathbf{r}_{NM}^P}_{f_N}\Big] \tag{11}$$

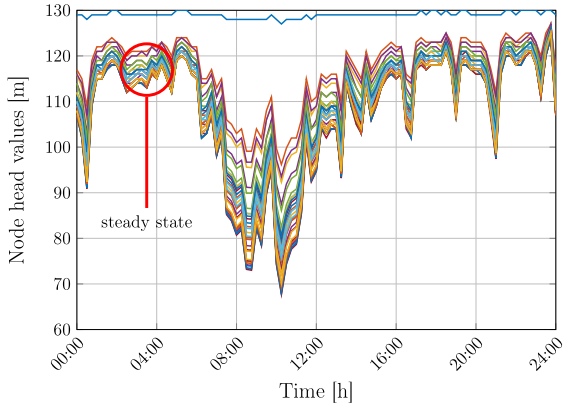
Remark 2. The residual ordering inside the matrix is not essential. It was chosen thus to make easier the grouping after fault occurrence (all cases which correspond to a certain node under fault are stacked consecutively). ♦

[5] With this notation, the nominal head, $\bar{\mathbf{h}}$, would be denoted as $\bar{\mathbf{h}}^{\bar{k}}$, where $p_{\bar{k}}$ is the profile active when the nominal head was measured. Since the nominal head remains constant (we cannot assume that $\bar{k}$ is updated), we keep the simpler notation $\bar{\mathbf{h}}$.

[6] Taking all possible combinations, the residual matrix has $D = N \cdot M \cdot P$ columns. For large-scale networks or if arbitrary selections of profiles, faults and fault magnitudes are considered, the value of D, and consequently, the arranging and content of $\mathbf{R}$, may differ.



(a) profile



(b) dynamical head response

Fig. 2. The head response for a given profile in the Hanoi network.

The Hanoi benchmark

To illustrate the aforementioned notions, we consider a often-used benchmark in the literature: the Hanoi water network [17]. As seen in Fig. 1, the network characteristics are: one tank and 31 junction nodes linked through 34 pipes (each with its own length and diameter); each junction node can be affected by a leakage and some of the nodes will have sensors mounted on them.

With the network profile (which multiplies each of the junction nodes' base demand) given in Fig. 2(a), we simulate the nodes' head under nominal functioning for one day (with 15 min sampling) through the EPANET software [6], as seen in Fig. 2(b). We observe that the empiric rule from Remark 1 holds: the head values remain steady around 3 AM, thus justifying the choice of constructing the head values with information from this time interval.

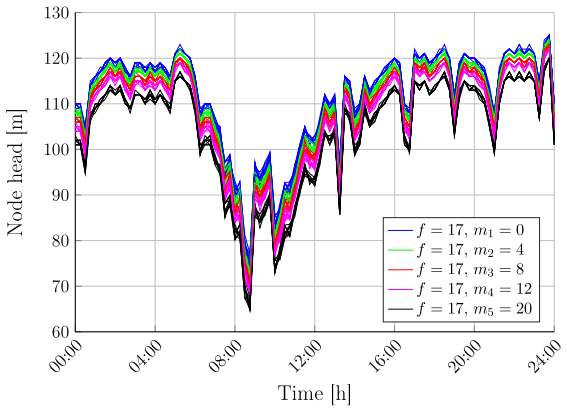


Fig. 3. Illustration of junction node 17 head variation while under fault and with various flow profiles and fault magnitudes.

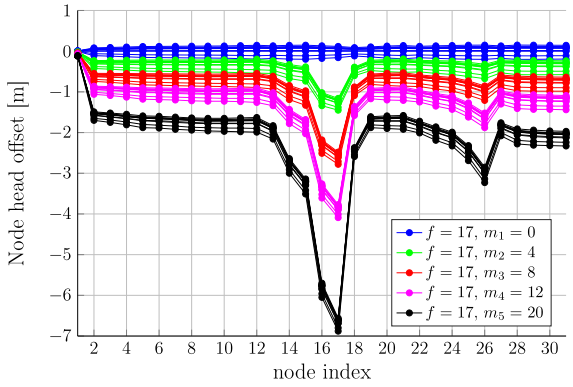


Fig. 4. Residual vectors in the Hanoi network.

Further, we consider 9 additional perturbations (hence $P = 10$) of the nominal profile shown in Fig. 2(a) through the addition of uniform noise bounded in the range of $\pm 2.5\%$. To illustrate the fault effects we consider such an event at node 17 with fault magnitudes taken from $\{0, 4, 8, 12, 20\}$, hence $M = 5$. Furthermore, for each fault magnitude we run $P = 10$ times the EPANET software (once for each flow profile). The resulting head values are shown (each group of plots with the same color denotes the node's value under the same fault magnitude but for different profiles) in Fig. 3 where we can observe, as expected, that the fault affects the node's head value.

Taking $K = [12, 18]$ and using it as in (9) to obtain the residuals (10) leads to the plots shown in Fig. 4 (we consider the absolute residual variant).

As expected, node 17 (where the fault happens) is the most affected. Still, measurable effects appear in nodes 14, 15 or 26. This is noteworthy for the subsequent fault detection/isolation analysis as it shows fault propagation throughout the network.

2.4. Problem statement

The main idea is to detect and isolate fault occurrences (i.e., leakages) within the water network with a limited amount of information (much fewer sensors than nodes). Due to its complexity (network size, nonlinearities, demand uncertainty, etc.), the problem is divided into two consecutive steps:

(i) the sensor placement, i.e., where to place the limited number of sensors such that the subsequent fault detection and isolation is maximized;

(ii) the fault detection and isolation procedure which provides an estimation of the fault occurrences (their location within the network).

The ideas, building on [12], are to provide a dictionary learning framework within which to:

(i) implement a Gram–Schmidt procedure which uses the gathered data to propose candidate nodes for sensor placement;

(ii) onto the reduced residual data, apply an online dictionary learning procedure which first trains a dictionary (an overcomplete basis for the residual signals) which is further used to classify test residuals into one of the predefined classes. Associating each class to a fault event means that the classification step itself becomes the fault detection and isolation mechanism.

Both elements exploit the network's structure and manage its numerical complexities: the network's Laplacian penalizes the sensor placement described in Section 3 and the online DL implementation described in Section 4 allows to handle large datasets (otherwise cumbersome or downright impossible through other, offline, procedures).

3. Sensor placement

Arguably, the main difficulty in maximizing the network's observability (and, thus, improve the FDI mechanism) comes from inadequate sensor placement: no subsequent FDI mechanism, (regardless of its prowess) can overcome the handicap of inadequate input data.

The problem reduces in finding a sequence of indices $\mathcal{I} \subset \{1 \dots N\}$ with at most s elements from within the list of available node indices such that the FDI mechanism provides the best results. As formulated, the problem has a two-layer structure: at the bottom, the FDI mechanism is designed for a certain sensor selection and at the top, the sensor selection is updated to reach an overall optimum. The nonlinearities and large scale of the problem mean that we have to break it into its constituent parts: first the sensors are placed (based on available data and/or model information) and, subsequently, the FDI mechanism is optimized, based on the already computed sensor placement.

While there are multiple approaches in the literature, the sensor placement problem is still largely open. One reason is that the degree to which a node is sensitive to a leak is roughly proportional with the inverse of its distance from the leaky node. Therefore, any selection strategy which does not use the entire information provided by the network is biased towards clumping sensor locations. On the other hand, analytic solutions which consider the network as a whole are computationally expensive (or downright impractical).

These shortcomings have motivated many works for various large-scale networks [18,19] as well as for water networks specifically [2,20,21]. While varied, the approaches can be grouped into [4]: (i) mixed-integer or greedy procedures which solve some variation of the set cover problem; (ii) evolutionary algorithms which employ heuristic methods; and (iii) topology-based methods which make use of the underlying graph structure of the network.

In particular, from the class of greedy procedures we analyze two minimum cover variants (set/test covering) and from the class of topology-based methods we propose an extended Gram–Schmidt procedure with Laplacian regularization.

3.1. Minimum cover procedures

Let us consider the residual matrix $\mathbf{R}$ defined as in[7] (11). To this matrix corresponds the *fault signature matrix* $\mathbf{M}$ obtained from the former by a binarization procedure [16]:

$$\mathbf{M}_{ij} = \begin{cases} 1, & \exists k: \text{ s.t. } k \text{ corresponds to fault } j \text{ and } |\mathbf{R}_{ik}| \geq \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

(12) should be read as follows: if any[8] of the entries of the ith node which correspond to the active fault j are above a pre-specified threshold τ then the fault j is detected by the node i (i.e., $\mathbf{M}_{ij} = 1$).

With $\mathbf{M}$, the fault signature matrix, given as in (12), we apply the *minimum set cover (MSC)* procedure from [2], in a variation of the mixed-integer form appearing in [4]:

$$\arg\max_{\gamma_j, \alpha_i} \quad \sum_{j \in \mathcal{F}} \gamma_j \tag{13a}$$

$$\text{s.t.} \sum_{i \in \mathcal{N}} \mathbf{M}_{ij} \alpha_i \geq \gamma_j, \; j \in \mathcal{F} \tag{13b}$$

$$\sum_{i \in \mathcal{N}} \alpha_i \leq s \tag{13c}$$

$$0 \leq \gamma_j \leq 1, \; j \in \mathcal{F} \tag{13d}$$

$$\alpha_i \in \{0, 1\}, \; i \in \mathcal{N}. \tag{13e}$$

$\mathcal{F}$, $\mathcal{N}$ denote (in our case, $\mathcal{F} = \mathcal{N} = \{1, \ldots, N\}$) the list of faults and nodes, respectively. Parameter s limits the number of available sensors. Taking $\gamma_j = 1$, (13) reduces to finding α_i such that (13b), (13c) and (13e) hold: (13b) ensures that each fault j is detected by at least a node i; (13c) ensures that at most s selections are made and (13e) ensures that the selection is unambiguous (a node is either selected, $\alpha_i = 1$ or not, $\alpha_i = 0$). This formulation may prove to be infeasible (there might be no node selection which permits complete fault detection), thus requiring the addition of the slack variables γ_j, their constraining in (13d) and subsequent penalization in the cost (13a).

As noted in [2], (13) maximizes fault detection but does not guarantee fault isolation (an ideal solution to (13) would be to find a unique node which detects all faults; this is, obviously, unfortunate from the viewpoint of fault isolation). The solution proposed in [2], at the cost of greatly expanding the problem size, is to construct an auxiliary matrix $\tilde{\mathbf{M}} \in \{0, 1\}^{|\mathcal{N}| \times \binom{|\mathcal{F}|}{2}}$:

$$\tilde{\mathbf{M}}_{i\ell(j_1,j_2)} = \mathbf{M}_{i,j_1} \cdot \mathbf{M}_{i,j_2}, \quad \forall j_1 \neq j_2, \text{ with } j_1, j_2 \in \mathcal{F}, \tag{14}$$

where $\ell(j_1, j_2)$ is an index enumerating all distinct unordered pairs (j_1, j_2). The idea is to construct an 'artificial' fault $\tilde{f}_{\ell(j_1,j_2)}$ and decide that node i is sensitive to it (i.e., $\tilde{\mathbf{M}}_{i\ell(j_1,j_2)} = 1$) iff only one of the faults happening at j_1 or j_2 is detected by node i. Replacing $\mathbf{M}$ with $\tilde{\mathbf{M}}$, defined as in (14), in (13) leads to the *minimum test cover (MTC)* procedure which maximizes fault isolation performance.

While other approaches exist in the literature, the MSC and MTC procedures presented above are representative in that they highlight some common issues which lead to a degradation of the subsequent FDI mechanism:

(i) Arguably, the application of a threshold as in (12) discards potentially useful information.

(ii) The sensor placement procedures are usually either model or data-based. Hybrid ones which make use of both are rarely encountered.

In the following subsection we propose an iterative method which combines unadulterated data (measured/simulated residual values corresponding to multiple fault occurrences) with model information (the graph structure of the network) to decide on an optimal sensor selection.

3.2. Graph-aware Gram Schmidt procedure

Recalling that $\mathcal{I} \subset \{1 \ldots N\}$ denotes the collection of sensor nodes indices, we note that the submatrix $\boldsymbol{R}_{\mathcal{I}} \in \mathbb{R}^{s \times D}$ will be the only data available for performing FDI. Thus, we want the low-rank $\boldsymbol{R}_{\mathcal{I}}$ matrix to approximate as best as possible the full-rank matrix $\boldsymbol{R}$. Further, if we look at sensor placement as an iterative process, then for each new sensor that we place we get access to the contents of one new row from $\boldsymbol{R}$.

Let $\boldsymbol{r}_i^\top$ denote row i of matrix $\boldsymbol{R}$. In order to achieve good matrix approximation we want to make sure that, when placing a new sensor in node i, the new row $\boldsymbol{r}_i^\top$ contains as much new information as possible about the water network. In other words, we want the projection of $\boldsymbol{r}_i^\top$ on the currently selected rows $\boldsymbol{R}_{\mathcal{I}}$ to be minimal:

$$i = \arg\min_{j \notin \mathcal{I}} \quad \left\| \text{proj}_{\boldsymbol{R}_{\mathcal{I}}} \boldsymbol{r}_j^\top \right\|_2. \tag{15}$$

In this context, the entire iterative process can be seen as a modified Gram–Schmidt orthogonalization process where we create a sequence of s orthogonal vectors chosen from a set of N, selected as in (15).

While the process induced by (15) might be good enough for matrix approximation, ignoring the water network's structure (even if implicitly present in the numerical data gathered in the residual matrix $\mathbf{R}$) is suboptimal.

Considering the underlying undirected weighted graph (via its Laplacian matrix), we are able to compute the shortest path between all nodes using Dijkstra's algorithm [22]. Thus, we update (15) to take into consideration the distances from the candidate node to the nodes from the existing set to encourage a better sensor placement spread across the network.

Let $\boldsymbol{\delta}_{\mathcal{I},j} \in \mathbb{R}^{|\mathcal{I}|}$ be the vector whose elements represent the distance between node j and each of the nodes from set $\mathcal{I}$. The penalized row selection criteria becomes

$$i = \arg\min_{j \notin \mathcal{I}} \quad \left\| \text{proj}_{\boldsymbol{R}_{\mathcal{I}}} \boldsymbol{r}_j^\top \right\|_2 + \lambda \sum_{i \in \mathcal{I}} \frac{1}{\delta_{i,j}} \tag{16}$$

where $\lambda \in \mathbb{R}$ is a scaling parameter. For $\lambda = 0$ (16) is equivalent to (15).

The penalty mechanism works as follows: if the projection is small and the sum of distances from node j to the nodes of $\mathcal{I}$ is large, then the distance penalty is small also and node j is a good candidate. On the other hand, if the sum of distances is small then the penalty grows and the possibility of selecting j decreases.

The result is a data topology-aware selection process, that encourages a good distribution of sensors inside the network in order to facilitate FDI. We gather the instructions necessary for sensor placement in Algorithm 1.

First, we select the row whose energy is largest (step 1) and place the first sensor there (step 2). We place the normalized row in the first column of matrix $\boldsymbol{U}$ (step 3) where we will continue to store the orthogonal vector sequence as discussed around (15). This auxiliary matrix helps us with future projections computations. From this initial state, step 4 loops until we place the remaining sensors. We begin iteration k by projecting the candidate rows $\mathcal{I}^c$ onto the existing selection (step 5). The $p_{i,j}$

[7] In fact, we use only a subset of columns from (11), the so-called *training set*, but for simplicity we abuse the notation.

[8] The "any" condition can be replaced with any selection criterion deemed necessary (e.g., "all entries", "the majority of the entries").

Algorithm 1: Gram–Schmidt Sensor Placement

Data: training residuals $\boldsymbol{R} \in \mathbb{R}^{N\times D}$,
shortest path between nodes $\boldsymbol{\Delta} \in \mathbb{R}^{N\times N}$,
sensors $s \in \mathbb{N}$,
distance penalty $\lambda \in \mathbb{R}$

Result: sensor nodes $\mathcal{I}$

1 Find dominant row: $i = \arg\max_k \left\|\boldsymbol{r}_k^\top\right\|_2,\ 1 \le k \le N$

2 Initial set: $\mathcal{I} = \{i\}$

3 Orthogonal rows: $\boldsymbol{U} = \left[\frac{\boldsymbol{r}_i^\top}{\left\|\boldsymbol{r}_i^\top\right\|}\right]$

4 **for** $k = 2$ **to** s **do**

5 Compute inner-products $\boldsymbol{P} = \boldsymbol{R}_{\mathcal{I}^c}(\boldsymbol{R}_{\mathcal{I}})^\top$

6 Project on the selection sub-space: $\boldsymbol{S} = \boldsymbol{P}\boldsymbol{U}^\top$

7 Compute projection norms:
$\boldsymbol{n} = \left[\left\|\boldsymbol{s}_1^\top\right\|_2 \quad \left\|\boldsymbol{s}_2^\top\right\|_2 \quad \cdots \quad \left\|\boldsymbol{s}_{n-k}^\top\right\|_2\right]$

8 Distance penalty: $n_j = n_j + \lambda \sum_{i\in\mathcal{I}} \delta_{i,j}^{-1},\ j \in \mathcal{I}^c$

9 $i = \arg\min n_j,\ j \in \mathcal{I}^c$

10 $\mathcal{I} = \mathcal{I} \cup \{i\}$

11 $\boldsymbol{u} = \boldsymbol{r}_i^\top - \boldsymbol{s}_i^\top$

12 $\boldsymbol{U} = \left[\boldsymbol{U} \quad \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}\right]$

13 **end**

element represents the projection of the candidate row i on the selected node j. Step 6 completes the projection by multiplying the inner-products with the corresponding orthogonal vectors. These two steps are required by (15) to compute the projection of all r_j^T on the selected rows $\mathcal{I}$. Next, we store in vector $\boldsymbol{n}$ the projection norm of each candidate row (step 7). We are given the shortest path between any two nodes in matrix $\boldsymbol{\Delta}$, where $\delta_{i,j}$ represents the distance from node i to node j. In step 8, we penalize the projections by summing up the inverse of the distance from candidate node j to each of the selected nodes. Node i corresponding to the smallest element in $\boldsymbol{n}$ is found (step 9) and added to the set $\mathcal{I}$. Steps 11 and 12 perform the Gram–Schmidt orthogonalization process: first the redundant information is removed from the row i (by subtracting the projection on the old set) and then the resulting vector is normalized and added to the orthogonal matrix $\boldsymbol{U}$.

Remark 3. The algorithm computations are dominated by the large matrix multiplications in steps 5 and 6. At step k, we need to perform $2Dk(N-k)$ operations in order to obtain the matrix $\boldsymbol{S}$. The rest of the instructions require minor computational efforts in comparison. This results in a complexity of $O(s^2ND)$ for the entire loop. ♦

Remark 4. Arguably, the weights appearing in the Laplacian graph should be proportional with the headloss between two linked nodes. This is not trivial since the headloss depends non-linearly on pipe length, diameter and roughness coefficient, see (3). ♦

Sensor placement in the Hanoi network

Using the example from Section 2, we now consider the three methods introduced earlier (MSC, MTC and Graph-GS) to generate sensor placements. We limit to the nominal profile case and compute the fault signature matrix $\mathbf{M}$ as in (12), for $\tau = 3$. The result is illustrated in Fig. 5 where a bullet at coordinates (i, j) means that the ith node detects the jth fault for at least one of its magnitudes. Arguably, instead of considering "any" we may

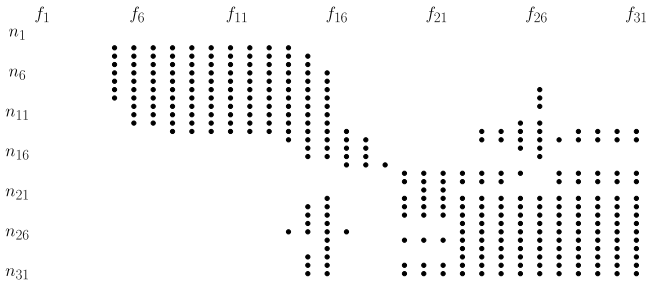


Fig. 5. Illustration of the fault signature matrix for the Hanoi water network.

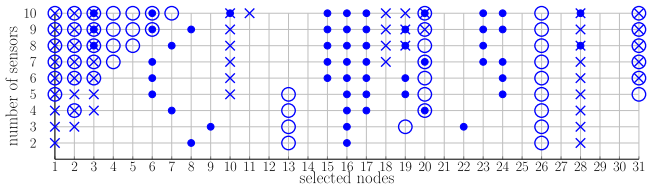


Fig. 6. Illustration of sensor selection for various methods.

consider that "all" or "most" of the fault magnitudes have to be detected at the node in order to consider it "fault-affected". Ultimately, classifying a node as being fault-affected is a mater of choice depending on the problem's particularities.

Applying the MSC procedure as in (13) leads to the sensor selection {1, 13, 20, 26, 31}. Constructing the extended signature matrix $\tilde{\mathbf{M}}$ as in (14) and using it for the MTC procedure leads to the sensor selection {6, 16, 17, 19, 24}. Lastly, the Graph-GS approach retrieves the selection {1, 2, 3, 10, 28} for the parameter $\lambda = 10^4$. In all cases, we assumed $s = 5$ (note that the MSC/MTC procedures may select fewer than s nodes since (13c) is an inequality).

The MSC and MTC procedures are able to run for this proof-of-concept network but the required number of binary variables increases in lock-step with the number of junction nodes for MSC and exponentially for MTC (e.g., in this particular example, 31 and, respectively, $\binom{31}{2} = 465$). The computation times are negligible here but they increase significantly in the case of large systems (as further seen in Section 5). Lastly, by counting the cases for which $\gamma_j = 0$ from (13) we estimate the number of fault detection errors in MSC (3 cases) and of fault isolation errors in MTC (34 cases). On the other hand, the Graph-GS procedure is much less sensitive to problem size and can handle easily large problems.

Fig. 6 illustrates the selections resulted for each method (circle, bullet and 'X' symbols for MTC, MSC and Graph-GS) for sensor numbers ranging from 2 to 10.

A word of caution is in order: regardless of method, the performance of a sensor selection is not truly validated until the FDI block is run and its output is compared with the actual fault occurrences. This is the scope of Section 4.

4. Dictionary learning and classification strategies

Dictionary learning (DL) [8] is an active field in the signal processing community with multiple applications such as de-noising, compression, super-resolution, and classification. Recent studies have also shown good results when dealing with anomaly detection in general [23–25], and particularly when applied to the FDI problem in water networks [12,13].

Dictionary learning

Starting from a sample $\boldsymbol{Y} \in \mathbb{R}^{s\times D}$, our aim is to find an overcomplete base $\boldsymbol{D} \in \mathbb{R}^{s\times B}$, called the dictionary, with whom we can represent the data by using only a few of its columns, also called atoms. Thus, we express the DL problem as

$$\min_{\boldsymbol{D},\boldsymbol{X}} \|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F^2 \tag{17a}$$

$$\text{s.t. } \|\boldsymbol{x}_\ell\|_0 \le s_0,\ \ell = 1:D \tag{17b}$$

$$\left\|\boldsymbol{d}_j\right\| = 1,\ j = 1:B, \tag{17c}$$

where $\boldsymbol{X} \in \mathbb{R}^{B\times D}$ are the sparse representations corresponding of the $\boldsymbol{Y}$ signals. (17b) dictates that each column $\boldsymbol{y}_\ell \in \boldsymbol{Y}$ has a sparse representation $\boldsymbol{x}_\ell$ that uses at most s atoms from $\boldsymbol{D}$ (i.e. $\boldsymbol{y}_\ell$ is modeled as the linear combination of at most s_0 columns from $\boldsymbol{D}$). (17c) is there to avoid the multiplication ambiguity between $\boldsymbol{D}$ and $\boldsymbol{X}$ (i.e., it allows to interpret the atoms as directions that the sparse representations follow. Thus, the elements in $\boldsymbol{X}$ act as scaling coefficients of these directions).

Remark 5. Solving (17) is difficult because the objective is non-convex and NP-hard. Existing methods approach the problem through iterative alternate optimization techniques. First the dictionary $\boldsymbol{D}$ is fixed and we find the representations $\boldsymbol{X}$, this is called the sparse representation phase and is usually solved by greedy algorithms among which Orthogonal Matching Pursuit (OMP) [26] is a fast, performant and popular solution. Next, we fix the representations and we find the dictionary. This is called the dictionary training or dictionary refinement phase and most algorithms solve it by updating each atom at a time (and sometimes also the representations using it) while fixing the rest of the dictionary. Popular routines are K-SVD [27] and Approximate K-SVD (AK-SVD) [28]. A few iterations of the representation and dictionary training steps usually bring us to a good solution. ♦

Dictionary classification

The choice of the s nonzero elements of a given column $\boldsymbol{x}_\ell$ (also called the representation support), highlights the participating atoms. These form a specific pattern which allows us to emit certain statements about the data which led to it. For example, let us assume that we can split the input data $\mathbf{Y}$ into distinct classes. Then, it follows naturally that signals from a certain class will probably use a characteristic pattern more than the signals from the other classes. Thus, we can classify a signal as being part of certain classes by just looking at the atoms used in its representation. In its direct form (17), the dictionary learning and classification suffers (at least from the viewpoint of FDI analysis) from a couple of shortcomings:

(i) the procedure is not discriminative: the learned dictionary may in fact lead to classifications with overlapping patterns of atom selection;
(ii) problem dimensions can grow fast as the number of network nodes increases.

Let c be the number of classes[9] and let us assume, without any loss of generality, that $\boldsymbol{Y}$ is sorted and can be split into c sub-matrices, each containing the data-items of a single class: $\boldsymbol{Y} = [\boldsymbol{Y}_1\ \boldsymbol{Y}_2\ \ldots\ \boldsymbol{Y}_c]$. An alternative method to (17) called Label consistent K-SVD (LC-KSVD) [9] adds extra penalties to (17) such that the atoms inherit class discriminative properties and, at the same time, trains a classifier $\boldsymbol{W}$ to be used afterwards, together with $\boldsymbol{D}$, to perform classification.

[9] Given that the data-items in $\boldsymbol{Y}$ are already labeled, we already know to which class they belong to.

Let $\boldsymbol{H} \in \mathbb{R}^{c\times D}$ be the data labeling matrix with its columns corresponding to the ones in $\boldsymbol{Y}$. If $\boldsymbol{y}_j$ belongs to class i then $\boldsymbol{h}_j = \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is the ith column of the identity matrix. Let matrix $\boldsymbol{Q} \in \mathbb{R}^{B\times D}$ be the discriminative matrix whose row numbers correspond to the dictionary atoms and whose column numbers correspond to the training signals. Column $\boldsymbol{q}_j$ has ones in the positions corresponding to the atoms associated with the class that $\boldsymbol{y}_j$ belongs to and zeros elsewhere. Usually atoms are split equally among classes. When the training signals are sorted in class order, matrix $\boldsymbol{Q}$ consists of rectangular blocks of ones arranged diagonally. LC-KSVD solves the optimization problem

$$\min_{\boldsymbol{D},\boldsymbol{X},\boldsymbol{W},\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F^2 + \alpha\,\|\boldsymbol{H} - \boldsymbol{W}\boldsymbol{X}\|_F^2 + \beta\,\|\boldsymbol{Q} - \boldsymbol{A}\boldsymbol{X}\|_F^2 \tag{18}$$

where the first term is the DL objective (17). The second term connects the label matrix $\boldsymbol{H}$ to the sparse representations $\boldsymbol{X}$ through matrix $\boldsymbol{W}$. We can view this as a separate DL problem where the training data are the labels and the dictionary is in fact a classifier. The trade-off between small representation error and accurate classification is tuned by parameter α. The third term learns $\boldsymbol{A}$ such that the linear transformation of $\boldsymbol{X}$ approximates $\boldsymbol{Q}$. Depending on the way we built $\boldsymbol{Q}$ (see above), this enforces that, first, only a few number of atoms can characterize a class and, second, these atom combinations cannot appear and are different from those for other classes.

After the learning process is over, in order to classify a signal $\boldsymbol{y}$, we need to first compute its representation with dictionary $\boldsymbol{D}$, again by using OMP or a similar algorithm, and then find the largest entry of $\boldsymbol{W}\boldsymbol{x}$ whose position j corresponds to the class that $\boldsymbol{y}$ belongs to

$$j = \arg\max_{i=1:c} (\boldsymbol{W}\boldsymbol{x})_i. \tag{19}$$

Remark 6. While (18) introduces additional variables it is, qualitatively, similar with (17) as it can be reduced to a similar formulation. Indeed, (18) can be reformulated as a "composite dictionary learning problem"

$$\min_{\boldsymbol{D},\boldsymbol{X},\boldsymbol{W},\boldsymbol{A}} \left\| \begin{bmatrix} \boldsymbol{Y} \\ \sqrt{\alpha}\boldsymbol{H} \\ \sqrt{\beta}\boldsymbol{Q} \end{bmatrix} - \begin{bmatrix} \boldsymbol{D} \\ \sqrt{\alpha}\boldsymbol{W} \\ \sqrt{\beta}\boldsymbol{A} \end{bmatrix} \boldsymbol{X} \right\|_F^2, \tag{20}$$

where $\mathbf{D}, \mathbf{W}, \mathbf{A}$ are learned from data provided by $\mathbf{Y}, \mathbf{H}$ and $\mathbf{Q}$, respectively. Note that in this particular instance, after the learning process $\boldsymbol{A}$ is discarded as it indirectly instilled discriminative properties to dictionary $\boldsymbol{D}$. ♦

4.1. Online learning

When dealing with large scale distribution networks the problem dimensions explode. Typical water networks may have hundreds or even thousands of nodes. Take as an example the case of a network with 5000 nodes. If each node represents one class, with 3 atoms per class we end up with a 15,000 column dictionary. Training on 30,000 signals, we end up with a 15,000 × 30,000 representations matrix. The computations involved become prohibitive on most systems. To accommodate large-scale scenarios we propose an online learning alternative.

Online DL handles one signal at a time, thus most operations become simple vector multiplications. At time t, we are given signal $\boldsymbol{y}$ which we use to update the current dictionaries $\boldsymbol{D}^{(t)}$, $\boldsymbol{W}^{(t)}$ and $\boldsymbol{Q}^{(t)}$. In order to speed-up results, a pre-training phase can be ensured where (18) is performed on a small batch of signals. The TODDLeR algorithm [10] adapts objective (18) for online learning using the recursive least squares approach [29]. Besides signal classification, its goal is to learn from all incoming signals: labeled or not. TODDLeR ensures that the model is not broken through

miss-classification by regulating the rate of change each signal brings

$$\min_{\boldsymbol{D},\boldsymbol{x},\boldsymbol{W},\boldsymbol{A}} \|\boldsymbol{y}-\boldsymbol{D}\boldsymbol{x}\|_2^2+\alpha\|\boldsymbol{h}-\boldsymbol{W}\boldsymbol{x}\|_2^2+\beta\|\boldsymbol{q}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda_1\|\boldsymbol{W}-\boldsymbol{W}_0\|_F^2+\lambda_2\|\boldsymbol{A}-\boldsymbol{A}_0\|_F^2. \tag{21}$$

For convenience, we dropped the (t) superscripts above. The problem does not have a closed-form solution and is solved in two steps. First, we solve the (18) problem for a single vector using the first three terms in (21). As shown in [10], this translates to updating $\boldsymbol{D}$, $\boldsymbol{W}$, $\boldsymbol{A}$ through a simple rank-1 update based on the $\boldsymbol{y}$ and its sparse representation $\boldsymbol{x}$. Keeping everything fixed in (21) except for $\boldsymbol{W}$ and $\boldsymbol{A}$ respectively, leads to the following two objectives

$$f(\boldsymbol{W})=\|\boldsymbol{h}-\boldsymbol{W}\boldsymbol{x}\|_2^2+\lambda_1\|\boldsymbol{W}-\boldsymbol{W}_0\|_F^2 \tag{22}$$

$$g(\boldsymbol{A})=\|\boldsymbol{q}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\lambda_2\|\boldsymbol{A}-\boldsymbol{A}_0\|_F^2 \tag{23}$$

meant to temper the updates brought by $\boldsymbol{y}$ to the dictionaries in the first step. Eqs. (22) and (23) are simple least-squares problems. By looking at f and g as generalized Tikhonov regularizations, it was motivated in [10] that good parameter choices are $\lambda_{1,2}=\|\boldsymbol{G}\|_2$ or $\lambda_1=\|\boldsymbol{W}_0\|_2$, $\lambda_2=\|\boldsymbol{A}_0\|_2$. Here $\boldsymbol{G}=\boldsymbol{X}\boldsymbol{X}^T$ is the Gram matrix of the pre-train representations that is then rank-1 updated in the online phase by each incoming signal $\boldsymbol{G}+\boldsymbol{x}\boldsymbol{x}^T$.

4.2. FDI mechanism

Recall that the DL procedure classifies an input vector wrt a set of a priori defined classes. Thus, assimilating a fault to a class means that the classification step of the DL procedure actually implements fault detection and isolation. The details are provided in Algorithm 2.

Algorithm 2: FDI Mechanism

Data: restriction of the residual matrix $\boldsymbol{R}_{\mathcal{I}}\in\mathbb{R}^{s\times D}$
Result: estimated class set $\hat{\mathcal{L}}$

1 partition indices into pre-train, train and test: $\mathcal{I}_{pt}\cup\mathcal{I}_{tr}\cup\mathcal{I}_{te}=\{1\dots D\}$, $\mathcal{I}_{pt}\cap\mathcal{I}_{tr}=\emptyset$, $\mathcal{I}_{pt}\cap\mathcal{I}_{te}=\emptyset$, $\mathcal{I}_{tr}\cap\mathcal{I}_{te}=\emptyset$
2 pre-train D, A, W as in (18) for labeled data $\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{pt}}$
3 **for** $r\in\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{pt}}$ **do**
4 update online D, A, W as in (21) for labeled residual vector $r\in\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{pt}}$
5 **end**
6 init the estimated class collection: $\hat{\mathcal{L}}=\emptyset$
7 **for** $r\in\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{te}}$ **do**
8 update online D, A, W as in (21) for unlabeled residual vector $r\in\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{te}}$
9 retrieve estimated active class $\hat{\ell}$ as in (19)
10 update the estimated class set $\hat{\mathcal{L}}=\hat{\mathcal{L}}\cup\{\hat{\ell}\}$
11 **end**

Considering the residual matrix $\mathbf{R}$ from (11) and the sensor selection $\mathcal{I}$ obtained in Section 3, we arrive to sub-matrix $\mathbf{R}_{\mathcal{I}}$. To this we associate the fault labels $\mathcal{L}$ (the active fault index for each of the columns of $\mathbf{R}_{\mathcal{I}}$).

Step 1 of the algorithm divides the residuals and associated labels into disjoint 'pre-train', 'train' and 'test' collections $\mathcal{I}_{pt}$, $\mathcal{I}_{tr}$, $\mathcal{I}_{te}$. These are used in steps 2, 4 and 8 to construct and respectively update the dictionary. Step 9 handles the actual FDI procedure by selecting the class best fitted to the current test residual ($r\in\mathbf{R}_{\mathcal{I}}^{\mathcal{I}_{te}}$). The class estimations are collected in $\hat{\mathcal{L}}$, at step 10 and compared with the actual test fault labels[10] $\mathcal{L}^{\mathcal{I}_{te}}$ to assess the success criterion of the FDI mechanism $S=(|\hat{\mathcal{L}}\cap\mathcal{L}^{\mathcal{I}_{te}}|/|\mathcal{L}^{\mathcal{I}_{te}}|)\cdot 100$.

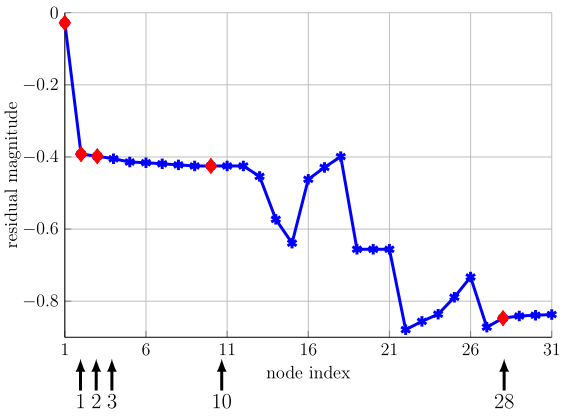


Fig. 7. Illustration of fault detection and isolation.

Remark 7. Arguably, it makes sense tweaking criterion S to count for near misses: the classification is successful not only if the correct node is identified but also if one of its neighbors is returned by the classification procedure. ♦

Remark 8. By construction, (19) returns the index corresponding to the largest value in the vector $\mathbf{Wx}$. This ignores the relative ranking of the classifiers, as it does not attach a degree of confidence for the selected index (i.e., large if there is a clear demarcation between classifications and small if the values are closely grouped. ♦

Illustration of the fdi mechanism

In our experiments each network node represents one class. During DL we used an extra shared dictionary to eliminate the commonalities within class-specific atoms [8]. This lead to $c=32$ classes for which we allocated 3 atoms per class leading to $n=3c$ dictionary atoms. An initial dictionary was obtained through pre-training on 2480 signals. Afterwards we performed online training on 2170 signals. Cross-validation showed good results when using $\alpha=4$ and $\beta=16$. When updating $\boldsymbol{W}$ and $\boldsymbol{A}$ we used $\lambda_{1,2}=\|\boldsymbol{G}\|_2$. With the resulting dictionaries we tested the FDI mechanism online on 4650 unlabeled signals. Applying the Graph-GS method for sensor selection, the rate of success was $S=80.09\%$.

For illustration, we show in Fig. 7 the full (blue line with bullet markers) residual signal corresponding to class 22 (i.e., the case where 22th node is under fault) and the actually-used data (red diamond markers), at nodes with sensors (those with indices from $\{1,2,3,10,28\}$). The actual classification was done as in (19) and resulted in a classifier vector $\mathbf{x}$ whose non-zero values are $\{7e-6,-1.2e-4,-7e-5,0.99,1e-3\}$, at indices $\{45,46,52,64,93\}$. Clearly, the most relevant atom in the description is the one with index 64 which lies in the subset $\{63,64,65\}$ corresponding to class 22. The classification $\boldsymbol{Wx}$ produces an indicator vector where the first and second largest

[10] The test fault labels are available since the analysis is carried out in simulation, in reality, the residuals obtained at runtime do not have such a label.

values are 0.999 and $1e-5$ thus showing that the procedure unambiguously produces the correct response (see Remark 8).

Further, we consider not only the success rate as defined in step 12 of Algorithm 2 but also count the cases where the fault is identified in the correct node's neighbors and in the neighbors' neighbors (as per Remark 7). This leads to an increase from 80, 09% to 90.69% and 98.92%, respectively, for the success rate.

Lastly, using the MSC sensor selection procedure we arrive to success rates 60.95%, 78.11% and 88.56% which proved to be significantly lower than the Graph-GS selection method. The MTC method, even for this small-scale network does not provide a solution.

Direct comparisons of results are somewhat difficult even when using the same network due to differences in flow profile, leakage values and the presence or absence of disturbances. As a qualitative indication of our relative performance we compare with the results of [17], that uses a well-accepted approach in model-based leak localization. Using a performance indicator (defined by eqs. (14, 17,18) in [17]) which increases linearly with the distance (in the topological sense) between the actual node under fault and the estimated one, we arrive at a value of 0.453, larger than the results shown, e.g., in Table 3 of [17] which range from 0.02 to 0.157. The results are reasonably close if we take into account our more realistic setup: more fault magnitudes (and some of them significantly smaller and hence harder to detect than the ones used in [17]) and the presence of noise-affected flow profiles (in [17] only a nominal profile is used). Note that the approach in [17] is using the hydraulic model of the network for leak localization, thus requiring much more information than the data-driven method proposed here. So, these similar leak localization results makes the proposed method competitive against model-based methods without requiring a detailed and well-calibrated model of the network.

From a computational viewpoint, our method is significantly better in the sensor selection step ([17] executes a semi-exhaustive search through a genetic algorithm whereas we solve an almost-instantaneous Gram–Schmidt procedure) and comparable in FDI performance (for us a quadratic optimization problem, for [17] a sensitivity matrix computation).

Further, we compare our FDI method with the standard linear learning methods from [30]. We do this both to compare with state-of-the art classifiers and because the methods from [17] do not scale reasonably for large-scale networks.

5. Validation over a generic large-scale water network[11]

To illustrate the DL-FDI mechanism, we test it over a large-scale generic network obtained via the EPANET plugin WaterNetGen [14]. To generate the network we considered 200 junctions, 1 tank and 253 pipes interconnecting them into a single cluster, as shown in Fig. 8.

5.1. Setup

To test our algorithms, we first take a nominal profile of node demands and perturb it with ±5% around its nominal values. Further, we consider that fault events are denoted by non-zero emitter[12] values in the EPANET emulator. With the notation from Algorithm 2 we run the EPANET software to obtain residual vectors (in absolute form) as follows:

(i) 2400 = 200 × 12 pre-train residuals; under the nominal profile, for each fault event we consider emitter values from the set of even values {8, 10, . . . , 30};

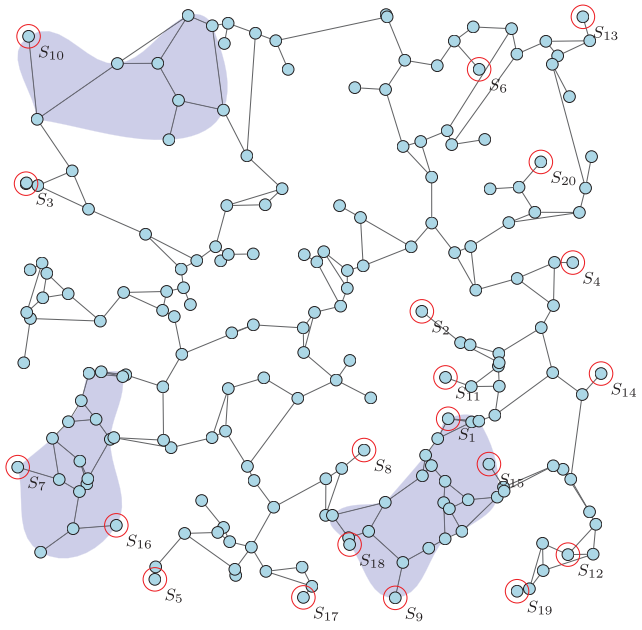


Fig. 8. Large-scale generic water network.

(ii) 2400 = 200 × 12 train residuals; for each node we consider 12 random combination of profile (from the set {1, 2, . . . , 10}) and of emitter value (from the set of odd values {9, 11, . . . , 31});

(iii) 3200 test residuals; we select random combinations of profile, fault and emitter values (taken from the sets {1, 2, . . . , 10}, {1, 2, . . . , 200} and {1, 2, . . . , 31}, respectively).

For further use we also divide the graph into 17 communities using the community detection tool [31]. For illustration we depict in Fig. 8 three of these (semi-transparent blue blobs).

The first step is to apply the sensor placement Algorithm 1 to retrieve the sub-matrix R_{pt} which gathers the pre-train residuals, taken at indices corresponding to sensor placements. The result is visible in Fig. 8 where we plotted (red circles) the first 20 sensor selections. Note that, due to the particularities of the Graph-GS method, each lower-order sensor selection is completely included within any larger-order sensor selection, e.g., selecting 5 sensors gives the collection {14, 21, 135, 142, 192} which is a subset of {14, 21, 41, 113, 117, 135, 137, 142, 170, 192}, obtained when selecting 10 sensors.

As a first validation we consider that each node fault is a distinct class and apply the DL-FDI mechanism described in Algorithm 2 to detect and isolate them. We quantify the success of the scheme in three ways, by counting all the cases where:

(S1) the estimated node equals the actual node under fault;

(S2) the estimated node is, at most, the neighbor of the node under fault[13];

(S3) the estimated node is, at most, the once-removed neighbor of the node under fault.

The three previous criteria can be interpreted as 0, 1 and 2-distances in the network's Laplacian. Arbitrary, n-distance, neighbors can be considered but their relevance becomes progressively less important.

The aforementioned community partitioning is another way of solving the FDI problem: each class corresponds to a community,

[11] Code available at https://github.com/pirofti/ddnet-online.

[12] In the EPANET software, to each junction node corresponds an emitter value which denotes the magnitude of the node leakage.

[13] Arguably this criterion imposes no performance penalty. The fault event is in fact a pipe leakage and associating the fault with a node is a simplification usually taken in the state of the art. In reality, if a node is labeled as being faulty, the surrounding ones need to be checked anyway.

i.e., any fault within the community is labeled as being part of the same class. This approach leads to an additional success criterion:

(S4) the estimated class corresponds to the community within which the fault appears.

In our simulations we set the parameters in (21) as follows: $\alpha = 4$ and $\beta = 16$ for the classification dictionaries as found via cross-validation [9,10]; for the update regularization we initialized $\lambda_{1,2} = 8$ and proceeded with automated parameter tuning $\lambda_{1,2} = \|\boldsymbol{G}\|_2$. TODDLeR was pre-trained, as earlier described, by first running LC-KSVD [9] on a small dataset in order to obtain an initial dictionary D and representations X. LC-KSVD used 20 iterations of AK-SVD [28] to train the atoms block belonging to each class and then 50 more iterations on the entire dictionary.

To assess our method we compare it with linear classifiers such as k-NN, SVM, and Naïve–Bayes, as commonly used for this type of problem (see, e.g., [30]). We mention that, even though these are also linear learning methods, they operate in bulk, by processing the entire dataset at once. This becomes prohibitive for large distribution networks and it is indeed the reason why we consider here an online method which provides reduced memory footprint and faster execution times.

5.2. Results

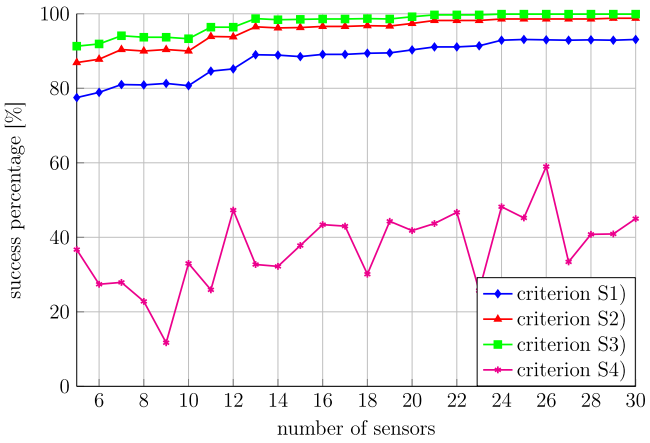
Running Algorithm 2 for a number of selected (as in Algorithm 1) sensors ranging from 5 to 30 we obtain the success rates shown in Fig. 9(a).

Several remarks can be drawn. First, and as expected, an increase in sensors, generally leads to an increase in performance. Still, care should be taken with the numbers considered: we note that even a small number (5) gives a good success rate and that after around $\geq$ 15 sensors the performance improvements taper off. Second, the classification failures appear to be 'near-misses' as can be seen when comparing the (S1), (S2) and (S3) criteria. The (S2) and (S3) values approach fast 100% which means that the difference (in the topological sense) between the estimated and the actual node under fault is small). In fact, having 24 or more sensors selected means that (as illustrated by the S3) criterion) the estimated fault location is never further away than 2 nodes from the actual fault location. Reducing the number of classes as in criterion (S4) significantly reduces the computation time but also leads to a marked decrease in performance (which does not appear to improve with an increase in the number of sensors). We consider this to be due to the uniform distribution of the nodes in the synthetic network considered here. We expect that applying this method in a typical residential water network (where dense neighborhoods are linked within the larger network by few long pipes) will provide better results.

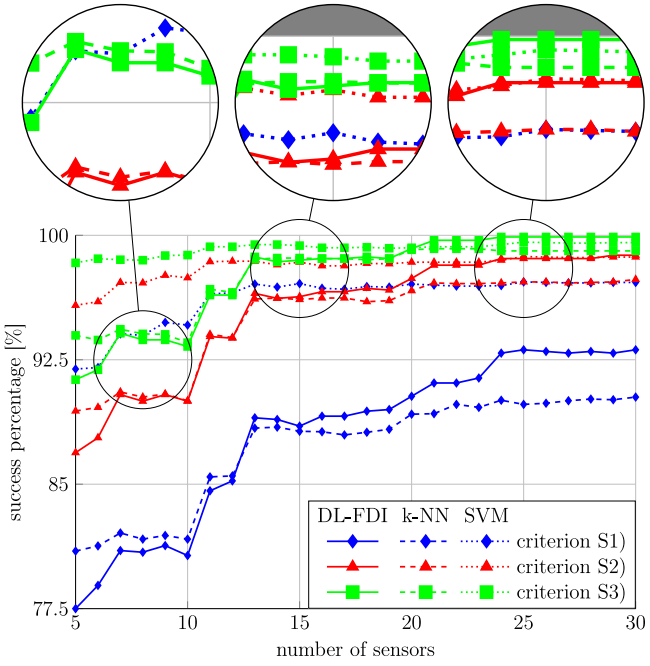
To assess our method we compare it with linear classifiers commonly used for this type of problem (see, e.g., [30]). Specifically, we consider the k-NN and SVM classifiers (with default Matlab settings, see implementation) and illustrate the results in Fig. 9(b): we consider criterions S1, 2 and 3) with blue diamond, red triangle and green square markers and plot the success rates for our implementation (DL-FDI), k-NN and SVM with solid, dashed and respectively, dotted lines.

Performance-wise, k-NN proves slightly better at small number of sensors but DL-FDI proves superior for larger values. SVM, on the other hand is better for small numbers of sensors and comparable with DL-FDI at larger values. Time-wise, k-NN is significantly more rapid (seconds) than DL-FDI (tens of seconds) and SVM is orders of magnitude slower than either of them (tens of minutes).

We note however that both k-NN and SVM consider all training data simultaneously and that the model they provide is fixed,



(a) DL-FDI method, criterion comparison



(b) DL-FDI versus k-NN and SVM classifiers

Fig. 9. FDI success rates for a large-scale water network.

whereas DL-FDI is implemented in its online form where residuals from the training and testing steps are fed to the algorithm one by one and the model is continuously updated. This means that k-NN and SVM are more sensitive to problem dimension and that at some point the memory requirements become prohibitive (which is not the case for our online DL-FDI implementation).

We also implemented the Bayesian method proposed in [30] but encountered numerical issues when using more than two sensors. In this method, each fault is associated a probability density map but it proved impractical to construct one from the available data.

5.3. DL-FDI in-depth analysis for 10 sensors

As stated earlier, the FDI is a classification procedure which exploits the discriminative and sparsity properties of the associated dictionary.

To highlight these properties we illustrate in Fig. 10 the active dictionary atoms obtained for the case of 10 sensors for each of

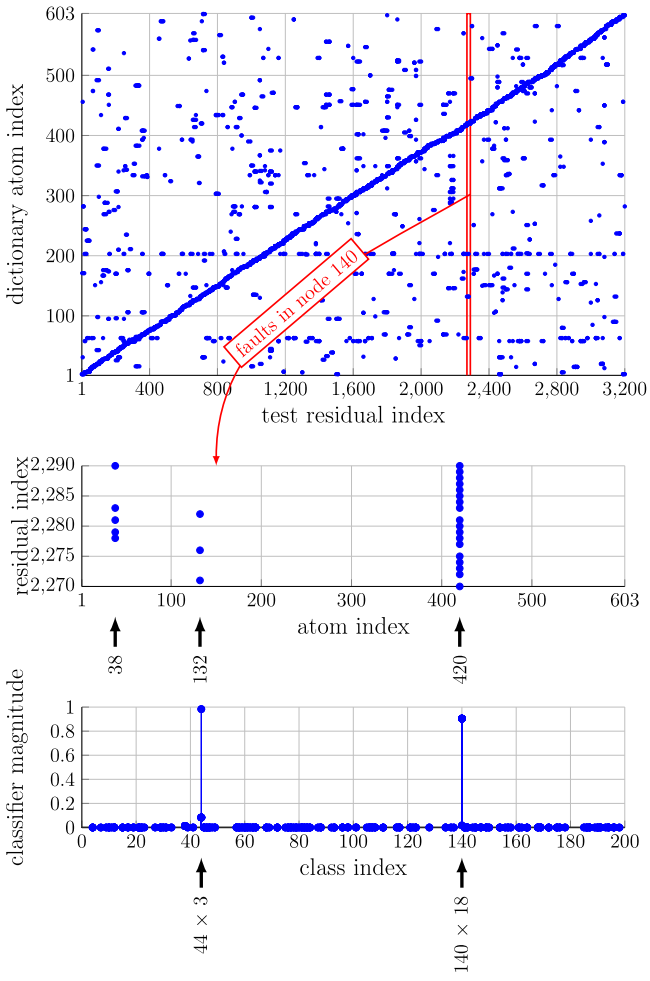


Fig. 10. Illustration of dictionary discrimination and sparsity properties (with detail for class 140).

the test residuals considered (a marker at coordinates (i,j) means that in the classification of the *i*th residual appears the *j*th atom). Note that for a better illustration we re-ordered the test residuals such that the faults appear contiguously.

To better illustrate the approach we take the test residuals corresponding to class 140 (faults affecting node 140), which in Fig. 10 correspond to residuals indexed from 2270 to 2290 and show them into the middle inset. We note that a reduced number of atoms ($\{38, 132, 420\}$) describe the residuals, hence proving the sparsity of the representation. The bottom inset plots the values of the classifier for each of the considered test residuals. We note that the classification returns 3 times class 44 (miss-classification) and 18 times class 140 (correct classification) – recall that, as per (19), the largest value in $\mathbf{Wx}$ indicates the class. For this particular class the success rates are around the average shown in Fig. 9(a), specifically, (S1) is $18 \cdot 100/21 = 85.71\%$, (S2) and (S3) are $21 \cdot 100/21 = 100\%$ since node 44 is the neighbor of node 140.

The diagonal effect in Fig. 10 is the result of how matrix $\boldsymbol{Q}$ was built. Recall that the lines in $\boldsymbol{Q}$ correspond to the atoms in $\boldsymbol{D}$ and its columns to the signals in $\boldsymbol{Y}$ and that we set element $\boldsymbol{q}_{ij}$ if atom i should represent signal j belonging to class c. This indirectly states that atom i has to represent signals of class c. The signals in Fig. 10 were resorted in class-order thus the atom index of the class-specific atoms (dictated by $\boldsymbol{Q}$) also changes every 20 or so residuals resulting in the ascending diagonal aspect. This is in fact the visual confirmation of the fact that our discrimination strategy worked as residuals might use atoms from the entire dictionary, but they always use at least one from their given class.

6. Conclusions

We have shown that data-driven approaches can be used successfully for sensor placement and subsequent fault detection and isolation in water networks. Performing sensor placement through a Gram-Schmidt-like procedure constrained by the network Laplacian and then using the resulting sensor data for online dictionary learning has allowed us to move forward from [12] and tackle large networks. Adaptive learning and classification [10] provides the benefit of a continuous integration of new data into the existing network model, be it for learning or testing purposes.

The results have shown good accuracy and pointed towards some promising directions of study such as: network partitioning into communities, adapting online dictionary learning to further integrate the network structure (e.g. by enforcing graph smoothness [32]) and providing synergy between the three phases: placement, learning, FDI (e.g. allow a flexible placement scheme where the learning iteration is allowed to change the sensor nodes based on current classification results).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] C. Zhao, H. Sun, Dynamic distributed monitoring strategy for large-scale nonstationary processes subject to frequently varying conditions under closed-loop control, IEEE Trans. Ind. Electron. 66 (6) (2018) 4749–4758.

[2] L.S. Perelman, W. Abbas, X. Koutsoukos, S. Amin, Sensor placement for fault location identification in water networks: A minimum test cover approach, Automatica 72 (2016) 166–176.

[3] J. Xu, P.S. Fischbeck, M.J. Small, J.M. VanBriesen, E. Casman, Identifying sets of key nodes for placing sensors in dynamic water distribution networks, J. Water Resour. Plan. Manag. 134 (4) (2008) 378–385.

[4] L. Sela, S. Amin, Robust sensor placement for pipeline monitoring: Mixed integer and greedy optimization, Adv. Eng. Inf. 36 (2018) 55–63.

[5] M. Brdys, B. Ulanicki, Operational control of water systems: Structures, algorithms and applications, Automatica 32 (11) (1996) 1619–1620.

[6] L.A. Rossman, EPANET 2: Users Manual, US Environmental Protection Agency. Office of Research and Development. National Risk Management Research Laboratory, 2000.

[7] R. Perez, G. Sanz, V. Puig, J. Quevedo, M.A.C. Escofet, F. Nejjari, J. Meseguer, G. Cembrano, J.M.M. Tur, R. Sarrate, Leak localization in water networks: a model-based methodology using pressure sensors applied to a real network in Barcelona [applications of control], IEEE Control Syst. 34 (4) (2014) 24–36.

[8] B. Dumitrescu, P. Irofti, Dictionary Learning Algorithms and Applications, Springer, ISBN: 978-3-319-78673-5, 2018, p. XIV, 284, http://dx.doi.org/10.1007/978-3-319-78674-2.

[9] Z. Jiang, Z. Lin, L. Davis, Label consistent K-SVD: Learning a discriminative dictionary for recognition, IEEE Trans. Pattern Anal. Mach. Intell. 35 (11) (2013) 2651–2664.

[10] P. Irofti, A. Băltoiu, Malware identification with dictionary learning, in: 2019 27th European Signal Processing Conference (EUSIPCO), IEEE, 2019, pp. 1–5.

[11] R. Jenatton, R. Gribonval, F. Bach, Local stability and robustness of sparse dictionary learning in the presence of noise, 2012, arXiv preprint arXiv:1210.0685.

[12] P. Irofti, F. Stoican, Dictionary learning strategies for sensor placement and leakage isolation in water networks, in: The 20th World Congress of the International Federation of Automatic Control, 2017, pp. 1589–1594.

[13] F. Stoican, P. Irofti, Aiding dictionary learning through multi-parametric sparse representation, Algorithms 12 (7) (2019) 131.

[14] J. Muranho, A. Ferreira, J. Sousa, A. Gomes, A. Sá Marques, WaterNetGen: an EPANET extension for automatic water distribution network models generation and pipe sizing, Water Sci. Technol.: Water Supply 12 (1) (2012) 117–123.

[15] G. Sanz Estapé, Demand Modeling for Water Networks Calibration and Leak Localization (Ph.D. thesis), Universitat Politècnica de Catalunya, 2016.
[16] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, J. Schröder, Diagnosis and Fault-Tolerant Control, vol. 2, Springer, 2006.
[17] M.V. Casillas, V. Puig, L.E. Garza-Castanón, A. Rosich, Optimal sensor placement for leak location in water distribution networks using genetic algorithms, Sensors 13 (11) (2013) 14984–15005.
[18] T. Kim, S.J. Wright, PMU Placement for line outage identification via multinomial logistic regression, IEEE Trans. Smart Grid 9 (1) (2018) 122–131.
[19] A. Krause, A. Singh, C. Guestrin, Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies, J. Mach. Learn. Res. 9 (Feb) (2008) 235–284.
[20] J. Meseguer, J.M. Mirats-Tur, G. Cembrano, V. Puig, J. Quevedo, R. Pérez, G. Sanz, D. Ibarra, A decision support system for on-line leakage localization, Environ. Model. Softw. 60 (2014) 331–345.
[21] T.T.T. Zan, H.B. Lim, K.-J. Wong, A.J. Whittle, B.-S. Lee, Event detection and localization in urban water distribution network, IEEE Sens. J. 14 (12) (2014) 4134–4142.
[22] E. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[23] A. Băltoiu, A. Pătraşcu, P. Irofti, Graph anomaly detection using dictionary learning, in: The 21st World Congress of the International Federation of Automatic Control, 2020, pp. 1–8.
[24] P. Irofti, A. Patrascu, A. Baltoiu, Fraud detection in networks: State-of-the-art, 2019, arXiv preprint arXiv:1910.11299.
[25] P. Irofti, A. Băltoiu, Unsupervised dictionary learning for anomaly detection, 2019, Arxiv: arXiv:2003.00293.
[26] Y. Pati, R. Rezaiifar, P. Krishnaprasad, Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition, in: 27th Asilomar Conf. Signals Systems Computers, vol. 1, 1993, pp. 40–44.
[27] M. Aharon, M. Elad, A. Bruckstein, K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation, IEEE Trans. Signal Proc. 54 (11) (2006) 4311–4322.
[28] R. Rubinstein, M. Zibulevsky, M. Elad, Efficient Implementation of the K-SVD Algorithm Using Batch Orthogonal Matching Pursuit, Tech. Rep. CS-2008-08, Technion Univ., Haifa, Israel, 2008.
[29] K. Skretting, K. Engan, Recursive least squares dictionary learning, IEEE Trans. Signal Proc. 58 (4) (2010) 2121–2130.
[30] A. Soldevila, R.M. Fernandez-Canti, J. Blesa, S. Tornil-Sin, V. Puig, Leak localization in water distribution networks using Bayesian classifiers, J. Process Control 55 (2017) 1–9.
[31] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.
[32] Y. Yankelevsky, M. Elad, Dual graph regularized dictionary learning, IEEE Trans. Signal Inf. Process. Netw. 2 (4) (2016) 611–624.