university of groningen

faculty of science and engineering

An algorithm for placing sensors ensuring fault detection and isolation in water distribution networks

Master Project Education Mathematics

July 2018

Student: I.W. Kauffeld

Studentnr: s2369702

First supervisor: Prof. dr. H.L. Trentelman

Second supervisor: Prof. dr. J. Top

Abstract

This thesis considers a fault detection and isolation problem for branched water distribution networks. From the literature [12] it is known that the problem of measuring contaminants in water distribution networks can be modeled by input-output systems in state-space form as well as using graph theory. It is shown that a graph topological condition for fault detection and isolation discussed in earlier research is also applicable to the water distribution networks. Using this new insight an algorithm was developed for the placement of sensors in water distribution networks. For known contamination locations in a network it is now possible to determine where sensors need to be placed to ensure fault detection and isolation.

Contents

1 Introduction

A water distribution network (WDN) provides safe drinking water to consumers in the right quantity. To provide safe drinking water, water distribution companies need a good real-time monitoring system [5]. Therefore, the water resources management community organized a design competition, namely the "Battle of the Water Sensor Networks (BWSN)" in 2006 [2]. This competition instigated a research interest and so research has been done on the topic of placing sensors in a WDN in the last decade.

This thesis also addresses the problem of placing sensors in WDNs. These sensors will be placed to help maintain good water quality as they will measure the concentration of contaminants present. Ideally, the sensors will be placed in such a way that it is possible to detect all contaminants, while also being able to determine where in the water network this contamination occurs. A problem such as the problem described can therefore be put in the category of fault detection and isolation (FDI) problems. Where in this particular case the faults are the presence of contaminants. This approach to the WDN problem has been already discussed in [12]. The result of that article is a necessary and sufficient condition for the solvability of the fault detection and isolation problem. Sharper conditions for solvability could exist and an algorithm, for placing sensors, incorporating the conditions for solvability has yet to be made. Water distribution companies can apply this algorithm to determine where in the network sensors need to be placed so that all contaminations can be detected.

The article [8] discusses linear dynamic FDI systems that can be defined over a graph. It presents a graph-theoretic condition ensuring fault detectability for networks that can be described by a simple, undirected and unweighted graph. In this thesis we will show that this graph-theoretic solvability condition can also be applied in the case of the WDN. This gives us insight into where in a WDN sensors need to be placed to ensure FDI solvability. The solvability condition is incorporated in an algorithm for placing sensors. This algorithm is implemented into Matlab to compute where in the network the sensors should be placed when the networks are too large to compute by hand. We will show results of the algorithm using Matlab for a few example networks.

First an explanation of the necessity of placing sensors in water distribution networks and the dynamics of water distribution networks are given. Then Section 3 reviews the available literature on sensor placement in WDNs and FDI. Section 4 discusses the main results of [8]. In Section 5 it is shown that the graph theoretic condition presented in [8] is also applicable to a water distribution network as described in Section 2 and what this means for a WDN. This graph-theoretic condition is then incorporated into an algorithm for placing sensors ensuring fault detection and isolation, which can be found in Section 6. Further, for a few example WDNs it will be computed where the sensors need to be placed using the algorithm in Matlab. Lastly, the conclusion of this thesis and recommendations for further research are discussed.

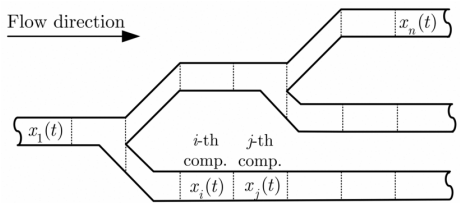

Figure 1: Simplified schematics of a WDN from [12]

2 Problem definition

The quality of drinking water is important as the distribution companies have to live up to certain standards according to World Health Organization (WHO), as well as other organizations (i.e. EU, USEPA) guidelines [5]. In addition, clean drinking water is a critical resource for the health and well-being of humans. To ensure the maximum allowable concentration of contaminants is not exceeded, a water distribution company performs periodic measurements of the water by taking samples. Unfortunately, with this method it may take days before a contamination is detected [3]. Sensors can be used to measure possible contaminants in real-time, which is less time consuming than taking samples. As these sensors are costly, the water distribution companies want a minimum amount of sensors in place while still ensuring contaminations are detected in time. Therefore, this problem can be studied as a model-based fault detection and isolation problem [12]. When the FDI problem is solved, it is possible to place the sensors such that their real-time measurements can be used to pinpoint where the contamination occurs in the network. In this way a water distribution company is able to take action against contaminations quickly and ensure safe drinking water.

2.1 Dynamics of water contamination in WDNs

Now an explanation of what is considered to be a WDN will be given, which corresponds with the definition presented in [12]. A water distribution network is considered to be a branched network, with no cycles, where the water is assumed to have a constant, laminar flow in one direction only. The total mass is conserved since only contaminants that do not interact with the environment, precipitate, nor re-suspend, once present in the network, are considered. The dynamics of the network are as follows and represented in terms of the concentration of one contaminant only. See Figure 1 for a simplified example of a WDN and an understanding of the dynamics of this network. A WDN is subdivided into $n \in \mathbb{N}^+$ different compartments with a constant volume $V_i \in \mathbb{R}^+$ (in m^3). The total number of compartments and the volume of each compartment are both finite. At a moment in time a compartment i has a certain concentration of contaminants $x_i(t) \in \mathbb{R}_0^+$ (in g/m^3). After a time interval $\Delta t \in \mathbb{R}^+$ (in s) the water, possibly containing contaminants, either stays in or leaves the compart-

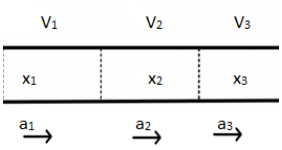

Figure 2: Pipe structure within a WDN

ment due to mass conservation. When the water leaves a compartment it flows directly into the compartment downstream with a constant water flow a_i (in m^3/s). For example this means that in Figure 1 the water from compartment i can only flow into compartment j.

There are three different structures possible in a WDN, which consist of different compartments. A compartment together with the adjacent compartments can either be part of a pipe, a junction or a conflux structure. Each of these structures has its own dynamics and will be discussed in further detail to be able to describe the dynamics of the whole network. In Figure 2 an example of a pipe structure within a WDN is given. The dynamics of this kind of structure in terms of compartment 2 are

$$V_2x_2(t+\Delta t) = V_2x_2(t) + a_1\Delta t x_1(t) - a_2\Delta t x_2(t)$$

Where $V_2x_2(t+\Delta t)$ stands for the amount of contaminant in compartment 2. $V_2x_2(t)$ represents the amount of contaminant present in compartment 2 at time t. $a_1\Delta t x_1(t)$ is the amount of contaminant that leaves compartment 2 over the timespan Δt. $a_2\Delta t x_2(t)$ is the amount of contaminant that enters compartment 2 over the timespan Δt. All amounts of contaminant are in g. In the same way one can state the equations defined over the compartments 1 and 3

$$V_1x_1(t+\Delta t) = V_1x_1(t) + a_1\Delta t x_1(t)$$

$$V_3x_3(t+\Delta t) = V_3x_3(t) + a_2\Delta t x_2(t) - a_3\Delta t x_3(t)$$

When one wants to write the aforementioned dynamics in a state space form, start with rearranging the terms. Then dividing by Δt, taking the limit as $\Delta t \to 0$. This results in

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} -a_1/V_1 & 0 & 0 \\ a_1/V_2 & -a_2/V_2 & 0 \\ 0 & a_2/V_2 & -a_3/V_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Notice that

$$\begin{pmatrix} -a_1/V_1 & 0 & 0 \\ a_1/V_2 & -a_2/V_2 & 0 \\ 0 & a_2/V_2 & -a_3/V_3 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 0 & 0 \\ a_1/V_2 & 0 & 0 \\ 0 & a_2/V_2 & 0 \end{pmatrix} + \begin{pmatrix} -a_1/V_1 & 0 & 0 \\ 0 & -a_2/V_2 & 0 \\ 0 & 0 & -a_3/V_3 \end{pmatrix}$$

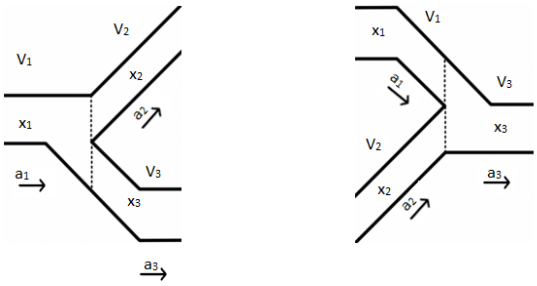

Figure 3: On the left a junction structure within a WDN and on the right a conflux structure

As one can see, the first matrix on the right of the latter equation is the adjacency matrix of the pipe structure in Figure 2. So the matrix representing the dynamics of the pipe structure can be split up in the adjacency matrix and a diagonal matrix as $A + D$.

The dynamics of the consecutive compartments 1, 2 and 3 from the left of Figure 3 are as follows

$$V_1 x_1(t + \Delta t) = V_1 x_1(t) + a_1 \Delta t x_1(t)$$

$$V_2 x_2(t + \Delta t) = V_2 x_2(t) + a_2 \Delta t x_1(t) - a_2 \Delta t x_2(t)$$
$$V_3 x_3(t + \Delta t) = V_3 x_3(t) + a_3 \Delta t x_1(t) - a_3 \Delta t x_3(t)$$

This yields in state space form

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} -a_1/V_1 & 0 & 0 \\ a_2/V_2 & -a_2/V_2 & 0 \\ 0 & a_3/V_3 & -a_3/V_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Once again the matrix representing the dynamics of a structure, in this case the junction, can be written as the adjacency matrix belonging to the graph of Figure 3 and a diagonal matrix as $A + D$.

The system dynamics of the conflux strucutre represented in Figure 3 is described by

$$V_1 x_1(t + \Delta t) = V_1 x_1(t) + a_1 \Delta t x_1(t)$$
$$V_2 x_2(t + \Delta t) = V_2 x_2(t) - a_2 \Delta t x_2(t)$$
$$V_3 x_3(t + \Delta t) = V_3 x_3(t) + a_1 \Delta t x_1(t) + a_2 \Delta t x_2(t) - a_3 \Delta t x_3(t)$$

Which in state space form this is represented by

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} -a_1/V_1 & 0 & 0 \\ 0 & -a_2/V_2 & 0 \\ a_1/V_3 & a_2/V_3 & -a_3/V_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Again the matrix representing the conflux dynamics of a WDN can be written as the adjacency matrix belonging to the graph of Figure 3 and a diagonal matrix.

A WDN can only consist of three possible structures, namely the pipe, the junction and the conflux structure. All three of these structures can always be modeled in state space form in terms of the adjacency matrix of the associated weighted graph and a suitable diagonal matrix as $A + D$.

Furthermore, at certain compartments there can be an inflow of contaminant: in a compartment where there is an inflow, the inflow is $f_k(t) \in \mathbb{R}_0^+$ (in g/m^3) with flow rate $b_k \in \mathbb{R}^+$ (in m^3/s), with $k = 1, \ldots, m$. In this context, these inflows of contaminant are considered to be the faults. Each fault, $f_k(t)$, is assumed to occur in only one compartment and only one fault per compartment can occur. Then the whole system dynamics yields

$$\dot{x}_i(t) = -\frac{a_i}{V_i} x_i(t) + \sum_{j \in N^-(i)} \frac{a_{ji}}{V_i} x_j(t) + \frac{f_l(t)}{V_i} \tag{1}$$

Where $N^-(i) := \{j \in 1, 2, \ldots, n \mid j \neq i,\ i \text{ succeeds } j\}$.

Finally, in certain compartments the concentration $x_j(t)$, with $j = 1, \ldots, p$, of contaminant can be measured using sensors. The whole state space model of the WDN is then given by

$$\Sigma : \begin{cases} \dot{x}(t) = (A + D)x(t) + Ff(t) \\ y(t) = Cx(t) \end{cases} \tag{2}$$

Where $x(t)$ is the state, $f(t)$ is the fault mode and $y(t)$ is the output vector. The matrix $A + D$ corresponds to the system dynamics of the WDN, without taking faults and sensors into consideration. The matrix F represents in which compartments fault can occur and finally the matrix C shows in which compartments measurements are taken. The system describing the dynamics of a WDN can be seen as a linear, time-invariant input-output system.

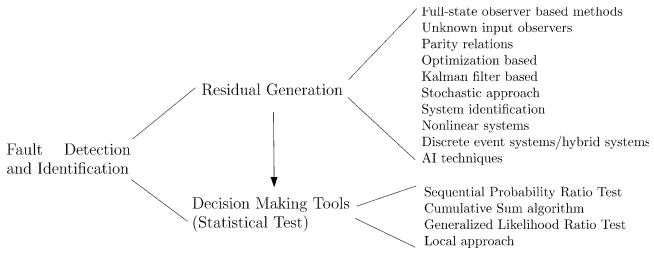

Figure 4: Overview of how FDI can be performed. Adopted from [4]

3 Literature review

We will now provide an overview of what has already been studied in the literature with regard to sensor placement in water distribution networks (WDNs), along with a discussion of different methods of solving fault detection and isolation problems.

As mentioned before, water distribution companies typically conduct manual sampling on a periodic basis and therefore it may take several days before a contamination is detected [3]. Hence, there is a need for real-time monitoring. However, specialized water quality sensors typically are expensive. In practice, this means that water distribution companies want to install as little sensors as possible while ensuring safe drinking water is supplied. This has motivated significant research in the field of placing sensors in water distribution network in the past years [3]. The most commonly applied strategies of sensor placement are based on the notion of contamination early warning systems [11]. These strategies aim to minimize the number of people affected in case of a contamination. This problem of placing sensors while ensuring that certain objectives are optimized and constraints are met within the network has been dealt with by several different research disciplines, such as operational research, combinatorial optimization, systems theory and control [2]. Several methodologies have thus been developed, but consensus among researchers about the objectives, methodologies and other aspects of sensor placement has not been achieved yet [9]. The notion of satisfying objectives and constraints is commonly done in the literature for sensor placement in WDNs. It is however different from the sensor placement algorithm developed in this thesis in the sense that the algorithm discussed here is based on a solvability condition for fault detection and isolation.

The approach of fault detection and isolation (FDI) has been extensively studied in the systems and control community, but is less common in the water community literature [12].The FDI problem consists of making a binary decision: either something has gone wrong or everything is fine, and then of determining the location of the fault if something has gone wrong [4]. A fault is thus to be understood as an unexpected change of the system function [7]. In the case of this thesis a fault is the presence of contaminants in the WDN. FDI can

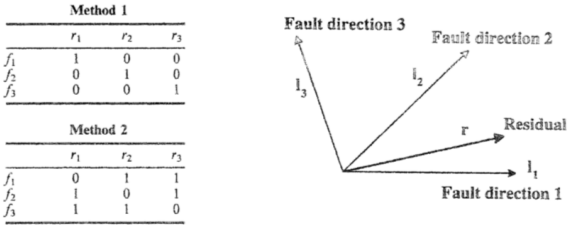

Figure 5: Different methods of residual generation. On the left structured residuals set and on the right fixed direction residual vector from [7]

either be model-based or data-based. Model-based FDI makes use of signals generated by the mathematical model representing the dynamics of the system. These signals are then compared with the actual measurements done by the system [7]. Data-based FDI uses measurements from redundant sensors [7]. Therefore, the advantage of model-based FDI is that no additional hardware is required for the implementation of the FDI system [7]. Faults in systems can be categorized into three different categories, namely actuator faults, sensor faults and component faults [4]. The contamination faults in the WDN belong to the actuator faults category and are therefore modeled as additive faults as is commonly done.

The solution to an FDI problem can be separated into two different steps [4]. The first step is to generate a set of variables known as residuals by designing one or more residual generating filters [4]. In the second step decisions on whether a fault has occurred need to be made [4], where in the ideal case the residual is zero if there is no fault and non-zero if a fault occurs. One also establishes which type of fault occurs in step two [4]. In the case of the water distribution network the latter corresponds with locating the faults in the network. Figure 4 provides an overview of all the different methods of executing the two steps of FDI.

Commonly, the residual represents the difference between the measured output and an estimated output [4]. In the literature two different ways of ensuring isolability of the faults is mentioned. The graphical representation of these methods in the case of three possible faults can be seen in Figure 5. One of the methods is called the structured residuals set, where each residual signal is designed to be sensitive to different faults (or subsets of faults) and thus be insensitive to all other possible faults [7]. The other method is that of a fixed direction residual vector. With this method a directional residual vector is designed to lie in a fixed and fault-specific direction (or subspace) [7]. A fault can then be isolated when the angle between the residual vector and the fault signature direction is very small [7].

4 FDI defined over graphs

In the following section the line of work from [8] will be discussed in detail. [8] starts with some geometric control definitions, followed by how geometric control theory is used to define the fault detection and isolation problem. One of the conditions needed to solve the fault detection and isolation problem is the need for a certain family of subspaces to be output separable. In the remainder of the article a sufficient condition based on graph-topological considerations for output separability is provided. This graph-topological condition for output separability can be used as a basis for an algorithm to place sensors in a WDN, which is the main problem of this thesis.

4.1 Geometric control theory

A geometric approach is used to state the control theory in coordinate-free form, which provides simpler results and gives insight into the actual meaning of statements [1]. Fault detection and isolation problems can be formulated using geometric language. Next the geometric control definitions needed for fault detection and isolation will be presented.

Consider a linear, time-invariant system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

Where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{p\times n}$. A subspace $\mathcal{S} \subseteq \mathbb{R}^n$ is then called (C, A)-invariant if

$$A(\mathcal{S} \cap ker(C)) \subseteq \mathcal{S}$$

This is equivalent to the existence of a $K \in \mathbb{R}^{n\times p}$ such that the subspace $\mathcal{S}$ is $(A + KC)$-invariant, meaning

$$(A + KC)\mathcal{S} \subseteq \mathcal{S}$$

Further, a family of subspaces $\{\mathcal{S}_i\}_{i=1}^q$ is called output separable if

$$C\mathcal{S}_i \cap (\sum_{j\neq i} C\mathcal{S}_j) = \{0\} \quad \text{for} \quad i = 1, 2, \ldots, q$$

4.2 Fault detection and isolation

Throughout [8], systems of the following form are considered

$$\Sigma : \begin{cases} \dot{x} = Xx + Mf \\ y = Nx \end{cases} \tag{3}$$

Where x is the state, f is the fault mode and y is the output vector. The matrices X, M and N are related to a given simple graph $G = (V, E)$, where

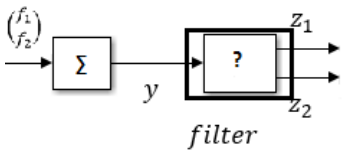

Figure 6: General FDI system

X belongs to a so-called qualitative class $\mathcal{Q}(G)$, M encodes the faulty vertices and N encodes the observer vertices.

Associated to a graph $G = (V, E)$, a family of matrices called the qualitative class of G is defined by

$$\mathcal{Q}(G) = \{X \in \mathbb{R}^{n\times n} \mid \text{for} \quad i \neq j, X_{i,j} \neq 0 \iff \{i, j\} \in E\}$$

Further, two subsets of V will play an important role and are denoted by V_F (faulty vertices) and V_O (observer vertices). Further, it is assumed that the first q vertices of the graph are faulty and the last s vertices are observed, so V_F and V_O are disjoint sets,

$$V_F = \{1, 2 \ldots, q\}$$

$$V_O = \{n - s + 1, n - s + 2, \ldots, n\}$$

Thus, the matrices M and N are as follows

$$M = \begin{pmatrix} I_q \\ 0_{n-q,q} \end{pmatrix} \text{and} \quad N = \begin{pmatrix} 0_{s,n-s} & I_s \end{pmatrix}$$

Figure 6 illustrates the general idea of a fault detection and isolation system in the case of two possible faults where the filter has yet to be designed. The goal of fault detection and isolation is that when one (or both) of the faults occur one should be able to see this change in the values of z_1 (or z_2 or both). To design a filter an observer is set up as follows

$$\Omega : \begin{cases} \dot{\hat{x}} = (X + KN)\hat{x} - Ky \\ \hat{y} = N\hat{x} \end{cases} \tag{4}$$

The observer problem is then to find a K such that $e(t) \to 0 (t \to \infty)$. In order to detect faults the error is defined as

$$e := \hat{x} - x$$

This satisfies the following dynamics

$$\dot{e} = (X + KN)e - Mf$$

$$r = \hat{y} - y = Ne$$

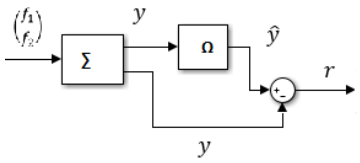

Figure 7: FDI after setting up an observer

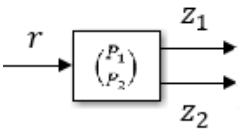

Figure 8: Final step FDI filter

Where r is the residual term. After setting up an observer the whole system thus far is represented in Figure 7.

In the simple case where only two faults are possible the following is the case if $e(0) = 0$

$$e(t) = \int_0^t e^{(X+KN)(t-s)} \begin{pmatrix} M_1 & M_2 \end{pmatrix} \begin{pmatrix} f_1(s) \\ f_2(s) \end{pmatrix} ds$$

If $f_1 \not\equiv 0$ and $f_2 \equiv 0$, R_1 is a linear combination of the columns of the matrix. $R_1 = \begin{pmatrix} M_1 & (X+KN)M_1 & \dots & (X+KN)^{n-1}M_1 \end{pmatrix}$. If $f_1 \equiv 0$ and $f_2 \not\equiv 0$, R_2 is a linear combination of the columns of the matrix. $R_2 = \begin{pmatrix} M_2 & (X+KN)M_2 & \dots & (X+KN)^{n-1}M_2 \end{pmatrix}$. R_1 and R_2 thus represent the reachable subspaces. We want to find the smallest $(X+KN)$-invariant subspace containing im M_1 and the smallest $(X+KN)$-invariant subspace containing im M_2: $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^n$ respectively. So the conditions to find these subspaces and a matrix K should be as follows

$$\begin{aligned} &1) \quad (X+KN)\mathcal{S}_1 \subseteq \mathcal{S}_1 \quad \text{and} \quad (X+KN)\mathcal{S}_2 \subseteq \mathcal{S}_2 \\ &2) \quad \text{im}M_1 \subseteq \mathcal{S}_1 \quad \text{and} \quad \text{im}M_2 \subseteq \mathcal{S}_2 \\ &3) \quad N\mathcal{S}_1 \cap N\mathcal{S}_2 = \{0\} \end{aligned}$$

Thus, $R_1 \subset \mathcal{S}_1$ and $R_2 \subset \mathcal{S}_2$. So the following also holds $NR_1 \cap NR_2 = \{0\}$. From this it can be concluded that R_1 and R_2 are independent. Figure 8 displays the final step of the FDI filter, where $P_1 : \mathbb{R}^n \to NR_1$ and $P_2 : \mathbb{R}^n \to NR_2$. P_1 and P_2 are projections along NR_2 and NR_1, respectively. If $f_1 \not\equiv 0$ and $f_2 \equiv 0$ is the case, the following holds due to the projections

$$e(t) \in R_1 \quad \forall t$$
$$r(t) = Ne(t) \in NR_1$$
$$z_2(t) = P_2 r(t) = 0$$
$$z_1(t) = P_2 r(t) \neq 0$$

z_1 and z_2 are the output of the whole FDI system. If fault f_1 occurs the corresponding output z_1 shows a change in value. Thus, the FDI system shows us by means of its output which fault must have occurred. The statements above illustrate the case if only two faults are possible. For the general case the conditions to find a family $\{\mathcal{S}_i\}_{i=1}^{q}$ of subspaces of $\mathbb{R}^n$ and associated matrix K are such that

$$1) \quad (X + KN)\mathcal{S}_i \subseteq \mathcal{S}_i, i = 1, \ldots, q \tag{5}$$
$$2) \quad \mathrm{im}M_i \subseteq \mathcal{S}_i, i = 1, \ldots, q \tag{6}$$
$$3) \quad N\mathcal{S}_i \cap \sum_{j \neq i} N\mathcal{S}_j = \{0\} \quad \text{for} \quad i = 1, 2, \ldots, q \tag{7}$$
$$4) \quad \sigma(X + KN) \subset \mathbb{C}_- \tag{8}$$

Equation (8) must be added to ensure the observer is asymptotically stable. As the observer is asymptotically stable if and only if all the eigenvalues of the matrix $X + KN$ have strictly negative real parts.

4.3 Graph-topological condition

Furthermore, a sufficient condition for output separability based on graph-topological considerations is given in [8]. Such an approach has the advantage of avoiding potential numerical and computational complexity issues associated with linear algebra computations for large-scale networks, providing instead robust conditions based on discrete mathematics. Moreover, it offers insight into the structural properties of networks with potentially useful applications in for example the design of network systems. The condition provided will be valid not only for a particular choice of the matrix X, but for a family of matrices within the qualitative class $\mathcal{Q}(G)$, namely the so-called distance-information preserving matrices.

For a graph $G = (V, E)$, the distance between two nodes is the length of the shortest path connecting them. The distance from node j to node i is denoted by $dist(j, i)$. By convention, $dist(j, i) := \infty$ if no path exists from node j to node i and $i \neq j$.

Definition 1 A matrix $X \in \mathbb{R}^{n \times n}$ is distance-information preserving with respect to the graph $G = (V, E)$ if

$$(X^k)_{i,j} \begin{cases} = 0 & \text{if} \quad dist(j, i) > k, \\ \neq 0 & \text{if} \quad dist(j, i) = k \end{cases}$$

for $k \geq 0$.

First a characterization of the subspaces $\mathcal{S}_i^*$ is presented. $\mathcal{S}_i^*$ is defined as the smallest (N, X)-invariant subspace containing imM_i for $i = 1, 2, \ldots, q$.

Lemma 2 Consider the system (3). Suppose that X is a distance-information preserving matrix with respect to the graph $G = (V, E)$. Let $i \in V_F$ and $d_i = dist(i, V_O)$. Then we have

$$\mathcal{S}_i^* = \text{im}\begin{pmatrix} M_i & X_i & \ldots & (X^{d_i})_i \end{pmatrix} \tag{9}$$

$$N\mathcal{S}_i^* = \text{im}N(X^{d_i})_i \tag{10}$$

Where X_i denotes the ith column of the matrix X. The proof can be found in [8].

The subspaces $\mathcal{S}_i^*$ are (N, X)-invariant and contain $\text{im}(M_i)$ by definition. So the conditions (5) and (6) of the geometric version of the fault detection and isolation problem discussed before are satisfied. Lemma 2 gives the necessary information to provide a graph-topological sufficient condition for the output separability condition (7). First, some necessary nomenclature is introduced.

A graph $H = (V, E)$ is bipartite if there exist disjoint vertex sets V^- and V^+ such that $V^- \cup V^+ = V$ and the edge set E contains only edges connecting one vertex from V^- and the other from V^+. With a slight abuse of notation, we write $H = (V^-, V^+, E)$ for bipartite graphs.

A set of t edges of a bipartite graph $H = (V^-, V^+, E)$ that do not share a common vertex is called a t-matching. A t-matching is called constrained if there is no other t-matching between the matched vertices.

Finally, given a simple graph $G = (V, E)$ and a pair (V_F, V_O), define $W_O = \{j \in V_O \mid dist(i, j) = dist(i, V_O) \quad \text{for some} \quad i \in V_F\}$. Thus, W_O consists of the observer vertices that are the closest to one of the faulty vertices. Define the bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$ with $\{j, i\} \in E_{OF} \iff j \in W_O, i \in V_F, dist(i, j) = dist(i, V_O)$. In other words, a path between a faulty node and an observer node is only represented as an edge in the bipartite graph if this path is equal in distance to the shortest path from the faulty node to one of the observer nodes.

Theorem 3 Consider the system (3) for a simple graph $G = (V, E)$ with faulty vertices V_F and observer vertices V_O. Then the family of subspaces $\{\mathcal{S}_i^*\}_{i=1}^q$ is output separable for any distance-information preserving matrix $X \in \mathcal{Q}(G)$ if the bipartite graph G_{OF} admits a constrained q-matching.

A proof can be found in [8].

The result of this theorem can be used for the design of systems, namely given a set of faulty vertices V_F, one way to guarantee output separability is to place sensors at certain (non-faulty) observer vertices so that the matching condition of Theorem 3 is satisfied.

5 Application of graph theoretic conditions

In this section we will show that Theorem 3 presented in [8] is also applicable to the problem defined in Section 2. As one recalls, Theorem 3 is proven in the case of a simple graph, $G = (V, E)$, where the matrix representing this graph has to be distance-information preserving (DIP). The application of Theorem 3 to the WDN problem will be proven by showing that Theorem 3 also holds for graphs representing a WDN and by proving that the matrix representing the system dynamics of the WDN is DIP.

5.1 Extension of Theorem 3

Theorem 3 was proven in [8] for a simple graph $G = (V, E)$, i.e. an undirected and unweighted graph containing no multiple edges or loops on nodes. In Section 2 it became clear that the graph of a WDN is a directed and weighted graph. Further, a WDN can always be described by an adjacency matrix A and an arbitrary diagonal matrix D. The presence of non-zero entries on the diagonal of a matrix representing a graph indicates that it may contain self-loops. Therefore, if we want to use Theorem 3 for the WDN problem there is a need to show that Theorem 3 can also be used for graphs that are directed, weighted and may contain self-loops.

Equation 10 is used to prove Theorem 3. Recall that the matrix N represents the observer nodes and $(X_i^d)_i$ the ith column of the matrix X, which represents fault i. After the multiplication of the matrix N with columns of the X matrix only paths from faults to observers remain. Hence, in Theorem 3 only paths with the considered length d_i from faults to observers are used to determine whether the bipartite graph G_{OF} admits a constrained q-matching are used.

The previous means that it does not matter whether or not there is also a path present from the observer node to the faulty node, as these entries of the DIP matrix are not used in the algorithm. Therefore, the DIP matrix could also be representing a directed graph and still yield the same outcome.

A graph can also be a weighted graph, as a weighted graph does not affect whether or not an entry is non-zero in the DIP matrix. The DIP matrix still gives a non-zero entry if there is a path, of the correct length, from a faulty node to an observer node. In the case of a weighted graph only the values of the entries differ, not the structure of the DIP matrix and the output separability of the subspaces.

Lastly, also self-loops do not change the outcome of the algorithm, as they do not affect the length of the path from a faulty node to an observer node. In a DIP matrix only the shortest path from one node to another is considered, so self-loops do not change a DIP matrix of a graph. Hence, Theorem 3 is also applicable to graphs that are directed, weighted and may contain self-loops.

5.2 $A + D$ is DIP

In order to prove that every $A+D$ matrix representing the dynamics of a WDN is DIP, the proof that every A matrix is DIP will be shown first.

Claim: Let $G = (V, E)$ be a simple graph and let A be its adjacency matrix. Then A is distance-information preserving.

Proof: By means of induction. For $k = 1$: $(A^k)_{i,j} = a_{i,j}$.
$Dist(j, i) = 1 \Rightarrow a_{i,j} \neq 0$ holds, since $a_{i,j} \neq 0$ if there is an edge between node j and node i according to the definition of an adjacency matrix.
$Dist(j, i) > 1 \Rightarrow a_{i,j} = 0$. This statement means there is no path of length 1 between node j and node i, and thus no edge. According to the definition of an adjacency matrix the entry of the matrix is zero if there is no edge present from node j to node i. Therefore, for $k = 1$ every adjacency matrix A is DIP.
Now the following is assumed to be true:

$$dist(j, i) = k \Rightarrow (A^k)_{i,j} \neq 0$$
$$dist(j, i) > k \Rightarrow (A^k)_{i,j} = 0$$

We need to prove that for $k = k + 1$ the matrix A is DIP. Let $B := (A^k)_{i,j}$.

$$\begin{aligned}
(A^{k+1})_{i,j} &= (A \times A^k)_{i,j} = (AB)_{i,j} \\
&= (i^{th} \text{ row of A}) \times (j^{th} \text{ column of B}) \\
&= \begin{pmatrix} a_{i,1} & a_{i,2} & \dots & a_{i,n} \end{pmatrix} \begin{pmatrix} b_{1,j} \\ b_{2,j} \\ \vdots \\ b_{n,j} \end{pmatrix} \\
&= a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \ldots + a_{i,n}b_{n,j} \\
&= a_{i,1}(A^k)_{1,j} + a_{i,2}(A^k)_{2,j} + \ldots + a_{i,n}(A^k)_{n,j}
\end{aligned}$$

$a_{i,1}(A^k)_{1,j}$ represents all the different paths with length $k + 1$ from node j to node i via node 1, since $a_{i,1}$ indicates whether there is an edge from node 1 to node i and $(A^k)_{1,j}$ has a value different from zero when there is a path of length k from node j to node 1, according to the assumptions above. So therefore $(A^{k+1})_{i,j} \neq 0$ when $dist(j, i) = k$, the first part of the definition for a matrix to be DIP holds.

For the second part of the definition for a matrix to be DIP, the value for $dist(j, i) > k + 1$ has to be checked. Based on the assumptions a $dist(j, i) > k$ means that there is no path of length k between node j and node i. $a_{i,1}$, $a_{i,2}$ until $a_{i,n}$ all represent a path of length 1. Thus, if $dist(j, i) > k+1$ then $(A^k)_{1,j}$, $(A^k)_{2,j}$ until $(A^k)_{n,j}$ have a distance $> k$. The values of $(A^k)_{1,j}$, $(A^k)_{2,j}$ until $(A^k)_{n,j}$ are then all zero. In conclusion, $(A^{k+1})_{i,j} = 0$ when $dist(j, i) > k$. The second part of the definition for a matrix to be DIP is thus true. Therefore,

every adjacency matrix A is DIP.

Claim Let $G = (V, E)$ be a directed and weighted graph and let A be its adjacency matrix. D is an arbitrary diagonal matrix. Then $A + D$ is distance-information preserving.

Proof: By means of induction. For $k = 1$, $dist(j, i) = 1$: $((A + D)^k)_{i,j} = ((A + D)^1)_{i,j} = (A^1)_{i,j}$ $(i \neq j) = a_{i,j} \neq 0$. The first condition for a matrix to be DIP is met. If $dist(j, i) > 1$: $((A + D)^1)_{i,j} \iff A^1_{i,j} = 0$. This has to be true since $A^1_{i,j} \neq 0$ would imply that $dist(j, i) = 1$, a contradiction.

The following is assumed to be true

$$dist(j, i) = k \Rightarrow ((A + D)^k)_{i,j} \neq 0$$

$$dist(j, i) > k \Rightarrow ((A + D)^k)_{i,j} = 0$$

We need to prove that for $k = k + 1$ matrix A is DIP.

$$((A + D)^{k+1})_{i,j} = ((A + D)(A + D)^k)_{i,j}$$

Let $C := (A + D)^k$, substituting this yields

$$\begin{aligned}((A + D)C)_{i,j} &= (AC + DC)_{i,j} \\ &= (AC)_{i,j} + (DC)_{i,j}\end{aligned}$$

Where

$$\begin{aligned}(AC)_{i,j} &= (i^{th} \text{ row of A}) \times (j^{th} \text{ column of C}) \\ &= \begin{pmatrix} a_{i,1} & a_{i,2} & \dots & a_{i,n} \end{pmatrix} \begin{pmatrix} c_{1,j} \\ c_{2,j} \\ \vdots \\ c_{n,j} \end{pmatrix} \\ &= a_{i,1}c_{1,j} + a_{i,2}c_{2,j} + \ldots + a_{i,n}c_{n,j} \\ &= a_{i,1}((A + D)^k)_{1,j} + a_{i,2}((A + D)^k)_{2,j} + \ldots + a_{i,n}((A + D)^k)_{n,j}\end{aligned}$$

and

$$\begin{aligned}(DC)_{i,j} &= (i^{th} \text{ row of D}) \times (j^{th} \text{ column of C}) \\ &= \begin{pmatrix} 0 & \dots & d_i & \dots & 0 \end{pmatrix} \begin{pmatrix} c_{1,j} \\ c_{2,j} \\ \vdots \\ c_{n,j} \end{pmatrix} \\ &= d_i c_{i,j}\end{aligned}$$

Thus,

$$((A+D)^{k+1})_{i,j} = a_{i,1}c_{1,j} + a_{i,2}c_{2,j} + \ldots + a_{i,n}c_{n,j} + d_i c_{i,j}$$

$a_{i,1}c_{1,j}$ represents the sum of all paths with length $k+1$ from node j to node i via node 1. $a_{i,1}$ represents the existence of an edge from node 1 to node i with length 1 and $c_{i,j}$ represents the sum of all paths with length k from node j to node 1.
$a_{i,2}c_{2,j}$ represents the sum of all paths with length $k+1$ from node j to node i via node 2. $a_{i,2}$ represents the presence of an edge from node 2 to node i with length 1 and $c_{i,j}$ represents the sum of all paths with length k from node j to node 2.
And so on for $a_{i,3}c_{3,j}$ until $a_{i,n}c_{n,j}$.

It was assumed that $dist(j,i) = k \Rightarrow ((A+D)^k)_{i,j} \neq 0$. Thus, there is at least one path with length k from node j to node i. It can then be deduced that if $dist(j,i) = k+1$ this means that there is at least one path of length $k+1$ from node j to node i. The values of at least one of the $c_{1,j}$, $c_{2,j}$ until $c_{n,j}$ has to be non-zero according to the assumption, as they represent $((A+D)^k)_{i,j}$ for a $dist(j,i)$ of k. $a_{1,j}$, $a_{2,j}$ until $a_{n,j}$ represent paths of length 1 and it was shown above that if there is an edge ($dist(j,i) = 1$) from a certain node to another node then $(A+D)^1_{i,j} \neq 0$. If $dist(j,i) = k+1$ there is a path of length $k+1$ from node j to node i, so at least one of the $a_{i,1}c_{1,j}$, $a_{i,2}c_{2,j}$ until $a_{i,n}c_{n,j}$ has a value different from zero. Furthermore, from the fact that $dist(j,i) = k+1$ it is clear that $c_{i,j} = 0$ due to the assumption of $dist(j,i) > k \Rightarrow ((A+D)^k)_{i,j} = 0$. So $dist(j,i) = k+1 \Rightarrow ((A+D)^{k+1})_{i,j} \neq 0$: the first condition for a matrix to be DIP holds.

Moving on to the case of $dist(j,i) > k+1$. A matrix $A+D$ is DIP if $dist(j,i) = k+1 \Rightarrow ((A+D)^{k+1})_{i,j} = 0$ is true. For this to be true all of the $a_{1,j}$, $a_{2,j}$ until $a_{n,j}$ represent a distance larger than 1 or all $c_{1,j}$, $c_{2,j}$ until $c_{n,j}$ represent a distance larger than k. If $a_{1,j}$, $a_{2,j}$ until $a_{n,j}$ represent a distance larger than 1 the values are zero according to what is shown above: for $k = 1$, $dist(j,i) > 1$ then $((A+D)^1)_{i,j} = 0$. If $c_{1,j}$, $c_{2,j}$ until $c_{n,j}$ represent a distance larger than k the values of $c_{1,j}$, $c_{2,j}$ until $c_{n,j}$ are also zero according to the assumption: $dist(j,i) > k \Rightarrow ((A+D)^k)_{i,j} = 0$. Since either $a_{i,1}$ or $c_{1,j}$ has a value equal to zero, one can see that $a_{i,1}c_{1,j}$ will always have a value equal to zero. This holds for all other representations $a_{i,2}c_{2,j}$ until $a_{i,n}c_{n,j}$. Thus, no path is present of length $k+1$ from node j to node i. If $c_{i,j}$ is not zero then there is a path: $dist(j,i) = k+1$, a contradiction. Thus, the value of $c_{i,j}$ has to be zero. Therefore every $A+D$ matrix is DIP.

5.3 Application of Theorem 3 to a WDN

The results of the article [8] can be applied to a water distribution network (WDN). From these results we conclude the following with regard to a WDN.

First, a compartment where an inflow of contaminant is possible cannot be used to place an observer as Theorem 3 demands that the graph G_{OF} is bipartite.

Second, in Section 2 it was stated that a WDN is defined as a network with no cycles. As the graph-theoretic condition in Theorem 3 also works for graphs that may contain cycles. The framework of possible WDNs on which the results from Theorem 3 can be applied can be extended to WDNs that may contain cycles.

Furthermore, the results of Theorem 3 can be used for the placement of sensors in a WDN. It is possible to guarantee output separability and thus fault detection and isolation, if one places sensors at certain (non-faulty) observer nodes in such a way that the constrained matching condition of Theorem 3 is satisfied. An additional note to this is, that in order for this constrained condition to hold there need to be at least as many observers as there are faulty nodes present.

Finally, the constrained matching ensures that the paths from faulty nodes to observer nodes are disjoint due to the formation of W_O and the definition of t-matching.

6 Algorithm

This section discusses which constraints need to be satisfied in order for a sensor placement algorithm to meet the condition of Theorem 3. Recall from 4 that V_F is a subset of V containing the faulty vertices, V_O a subset of V containing the observer vertices and W_O consists of the observer vertices that are the closest to one of the faulty vertices. A set of t edges of the bipartite graph is called a t-matching if these edges do not share a common vertex. Given a water distribution network and its associated graph $G = (V, E)$ where V_F is given and the algorithm will determine V_O. The algorithm needs to take into account that the observers should be placed in a way that ensures fault detection and isolation. From Theorem 3 and the previous chapter we conclude that this is possible if the bipartite graph G_{OF} admits a constrained q-matching.

First, an algorithm was developed for a q-matching to exist between the faults and the observers in the bipartite graph G_{OF}. Subsequently, this algorithm was extended such that the complete algorithm will ensure the found q-matching (if one exists) will be a unique matching.

6.1 Matching algorithm

To ensure a matching (if one exists) between the faulty nodes and the observer nodes the following conditions need to be satisfied

$$\forall i \in V_F \quad \exists j \in V_O : \quad \text{there is a path from} \quad i \quad \text{to} \quad j \tag{11}$$

$$\forall i \in V_F \quad \exists j \in V_O : \quad dist(i, j) = dist(i, W_O) \tag{12}$$

The algorithm must satisfy condition (11) in order for every faulty node i to have at least one path to observer node j. Condition (12) ensures that every faulty node i also has a path to an observer contained in W_O. Since the goal is to find a q-matching, where q is the number of faulty nodes, additional conditions are needed.

As the water companies want to install as few observers as possible, the algorithm should place the minimum amount of observers necessary to ensure fault detection and isolation. The algorithm is thus developed such that the algorithm places as many observers as there are faults possible.

The steps of the algorithm are the following:

Step 1 Choose o_1 in such a way that there is a path from node f_1 to node o_1.
Step 2 Choose o_2 such that the following constraints are met:
- $dist(f_1, o_2) \geq dist(f_1, o_1)$
- $dist(f_2, o_1) \geq dist(f_2, o_2)$
- There has to be a path from node f_2 to node o_2

Step 3 Choose o_3 such that the following constraints are met:
- $dist(f_1, o_3) \geq dist(f_1, o_1)$

- $dist(f_2, o_3) \geq dist(f_2, o_2)$
- $dist(f_3, o_1) \geq dist(f_3, o_3)$
- $dist(f_3, o_2) \geq dist(f_3, o_3)$
- There has to be a path from node f_3 to node o_3

And so on. In each step an observer is assigned, thus q steps are necessary.

Where f_1 is the first node in V_F, f_2 the second node in V_F and so on and o_1 the first node in V_O, o_2 the second node in V_O and so on.

Let us explain the constraints for step 2. The constraint $dist(f_1, o_2) \geq dist(f_1, o_1)$ ensures that the path from the fault already treated to the new observer is equal to or longer than the path between the first fault and the first observer. In this way the algorithm can guarantee that the edge between the first fault and first observer exists in the bipartite graph G_{OF} of the first two steps. In each step of the algorithm there are thus constraints ensuring an edge in the bipartite graph between the first fault and first observer, between the second fault and second observer and so on. The constraint $dist(f_2, o_1) \geq dist(f_2, o_2)$ combined with the constraint that there has to be a path from the second fault to the second observer ensures that the second fault will have an edge in the bipartite graph to the second observer. In this way all the constraints of each step together ensure that every fault i has an edge in the bipartite graph to observer i. Thus the algorithm determines V_O such that $V_O = W_O$.

6.2 Unique matching algorithm

For the algorithm to provide us with a unique matching additional constraints are necessary. One way for the algorithm to generate a constrained matching is to alter the constraints regarding the distances between faults and observers to constraints where the distances on the left hand side have to be strictly larger than the distances on the right hand side of the constraint. Step 2 of the algorithm will then be the following

Step 2 Choose o_2 such that the following constraints are met:
- $dist(f_1, o_2) > dist(f_1, o_1)$
- $dist(f_2, o_1) > dist(f_2, o_2)$
- There has to be a path from node f_2 to node o_2

With the new constraints every faulty node and every observer node has a degree exactly equal to 1, where the degree of a vertex is the numbers of edges incident to this vertex. The downside of this method is the chance that the algorithm is not able to find a q-matching is increased. Some possibilities to place an observer will not be taken into account. If no matching is found this does not necessarily mean that fault detection and isolation cannot be achieved. An approach that can check whether or not a found matching is unique and is also able to choose a different observer if the matching is not constrained is desired. In this way all possibilities to place the observers are checked before the algorithm end by stating that it is not possible to find a constrained matching.

The different approach starts with the steps described in Section 6.1 and then admits a bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$ to check whether the existing matching is unique. The bipartite graph G_{OF} is constructed from the graph associated to the WDN, where the nodes of the bipartite graph G_{OF} are all the faulty and observer nodes from the graph of the WDN. An undirected edge is present in the graph G_{OF} if a path exists in the WDN graph from faulty node i to observer node i, additionally this path must have a distance equal to the shortest distance from this faulty node i to one of the observers in W_O. There is a matching in the bipartite graph G_{OF} if each faulty and observer node has a degree ≥ 1. The matching is constrained if there is only one matching possible. The next step of the algorithm is:

Step $q+1$ Construct bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$

After step $q+1$ the algorithm should check if the bipartite graph contains more than one matching. According to [10] a matching M of an undirected bipartite graph G is an edge set such that no two edges of M share their endpoints. This corresponds with the definition that is given for t-matching in [8]. For fault detection and isolation it is necessary that every faulty and observer node has a degree ≥ 1. In [10] this complies with the definition of a perfect matching: a matching M is perfect if all vertices of G are incident to some edges of M. For the matching to be constrained in terms of [8] the bipartite graph G_{OF} may contain only one perfect matching.

One of the properties of perfect matchings in undirected bipartite graphs is that there exists another perfect matching if and only if there exists an alternating cycle [10]. Therefore, the algorithm can check whether or not the bipartite graph contains an alternating cycle, since only one perfect matching exists if no alternating cycles are present. A cycle C is an alternating cycle if two edges in $C \backslash M$ are not adjacent, thus edges of M and edges that are not in M then appear alternatively [10]. Due to the mathematical conditions already stated there exists at least one perfect matching and this perfect matching is also known. The constraints of the algorithm always ensure that fault i ($i = 1, \ldots, q$) has a path to the ith placed observer. Figure 9 displays an example of a bipartite graph showing the perfect matching that will always be generated by the algorithm (if a q-matching exists). The next step of the algorithm is:

Step $q+2$ Check if bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$ has alternating cycles

The algorithm checks whether the matching found is unique after all observers are assigned to a node. Unfortunately, it cannot be retraced after which step the matching is no longer unique. Once the matching generated by the algorithm is not unique anymore (after placing a certain observer) it will never be a unique matching after placing more observers as the cycle in the matching remains. To prevent that all the steps of the algorithm have to be executed multiple times until a perfect matching is found the algorithm is altered. With the alteration,

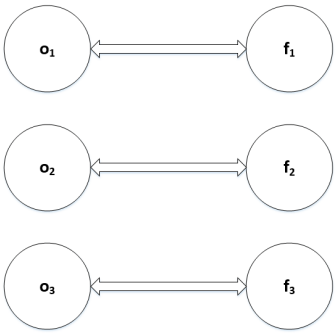

Figure 9: Example of a bipartite graph G_{OF} with three observers and three faults

each time an observer is assigned a check is done whether or not the matching until then is unique. If the matching is no longer unique the last executed step will be redone and thus a different observer will be assigned. If all the possible observer options by redoing step i cannot result in a unique matching then step $i-1$ will be reiterated. The final algorithm is the following

Step 1 Choose o_1 in such a way that there is a path from node f_1 to node o_1.
Step 2.1 Choose o_2 such that the following conditions are met:
- $dist(f_1, o_2) \geq dist(f_1, o_1)$
- $dist(f_2, o_1) \geq dist(f_2, o_2)$
- There has to be a path from node f_2 to node o_2

Step 2.2 Make bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$
Step 2.3 Check whether bipartite graph $G_{OF} = (W_O, V_F, E_{OF})$ has alternating cycles
if G_{OF} contains alternating cycles and a different o_2 cannot be chosen **then** choose a different o_1
if G_{OF} contains alternating cycles **then** choose a different o_2
And so on for step 3 until q.

For small networks it is possible to perform the algorithm by hand, but for larger networks this would be increasingly difficult and time consuming. Therefore, the algorithm was implemented into Matlab. The Matlab code can be found in the appendix. The next chapter will treat some examples of water distribution networks with known faulty nodes for which the algorithm can provide a list of nodes that should be observer nodes using Matlab.

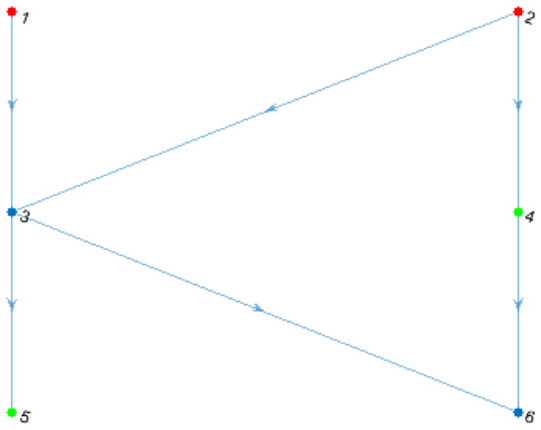

Figure 10: Example 1 of a WDN, where a red node is a faulty node and a green node an observer node

7 Examples using Matlab

First, two examples of small networks will be treated to illustrate how the algorithm works. After this an example of a larger water distribution network is discussed.

Figure 10 displays the first example of a possible water distribution network. In reality these networks will be much larger than the one displayed, but for our purposes this suffices. This example has six nodes in total of which two are labeled as faulty, which in this case are node 1 and node 2. After running the algorithm in Matlab, the result is displayed in Figure 10. Matlab returns the original WDN graph in which the nodes to be assigned observers are highlighted with a green color. The second part of the output is an array containing the nodes that should be observers. The time Matlab needs to compute this is approximately 2.7 seconds.

Step by step the algorithm works as follows: first, it is determined which nodes meet the constraints. The Matlab code is arranged in such a way that the possible observer node that is the farthest away from the faulty node will be chosen as the observer node. Thus, in the case of our example: node 5 is chosen. Matlab then determines which nodes could be observers for the second fault: node 6 is chosen. Subsequently, Matlab constructs a bipartite graph and checks whether this graph contains cycles: this seems to be the case. If the bipartite graph contains cycles Matlab examines whether this cycle is an alternating cycle. Multiple perfect matchings are present if the bipartite graph contains alternating cycles. The constructed bipartite graph appears to contain an alternating cycle meaning that the matching is not constrained. Step 2

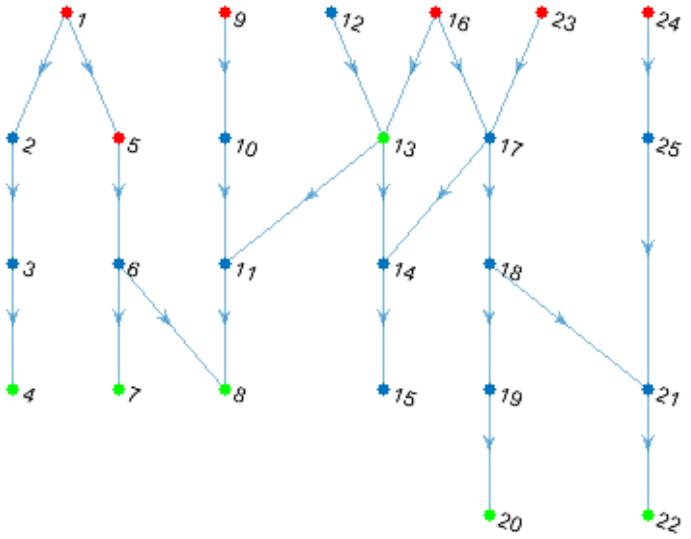

Figure 11: Example 2 of a WDN, where a red node is a faulty node and a green node an observer node

needs to be redone. Assigning node 6 in step 2 did not result in a constrained matching, so first node 6 is removed from the list of nodes that could be possible observers for the second fault. Node 4 is chosen in redoing step 2 and a check whether the bipartite graph contains cycles is performed: no cycles exist. The matching is thus constrained and Matlab provides us with a graph of the WDN in which we can see which nodes are faulty and which should be observers.

The next example of a WDN is displayed in Figure 11 and is an example of a network for which it is necessary for the algorithm to redo step i and step $i-1$. Nodes 4, 7 and 8 are chosen as the observers in the first three steps and those steps do not need to be redone in the remainder of the algorithm. In step 4 and 5 of the algorithm nodes 15 and 21 are chosen respectively. It turns out that the matching is not a constrained matching and step 5 will be redone. Step 5 is redone a number of times until no possible observers are left to choose from: the algorithm then needs to redo step 4 as well. In redoing step 4 the observer that was first chosen will be removed from the list of possible observer nodes. Step 4 and step 5 are redone a number of times until a matching is found that is unique again. The time Matlab needs to compute the result presented on the right of Figure 11 is approximately 2.8 seconds.

The last example network treated, displayed in Figure 12, is an example of a network that is a bit larger than the previous examples. In addition to the nodes displayed in Figure 12, node 1 is also a faulty node. When submitting the list of faulty nodes including node 1 the output Matlab gives is the following: No constrained matching possible with first observer at node 27. The time Matlab

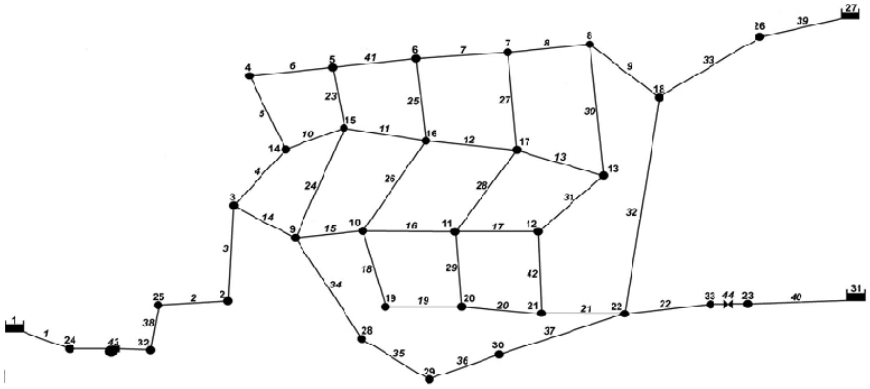

Figure 12: Example 3 of a WDN (adopted from [6])

needs to determine this is approximately 2 seconds. Note that the Matlab code presented in the Appendix is not able to redo step 1. If the algorithm generates printed text as output stating that it is not possible to find a matching by placing the first observer at a certain node it is needed to manually alter the class of the node such that these nodes cannot be chosen as the first observer. In the case of this example we can see in Figure 13 that if we place the first observer at node 3 for example, this will not result in multiple perfect matchings. Therefore, node 1 is removed from the list of faulty nodes and Matlab computes for us which nodes should also be assigned to be observers to ensure fault detection and isolation. The result is shown in Figure 13 and Matlab needs approximately 2.5 seconds to give this result. The computing time is relatively short, thus even in the case of larger water distribution networks it is likely the time it will take Matlab to run the algorithm will be acceptable.

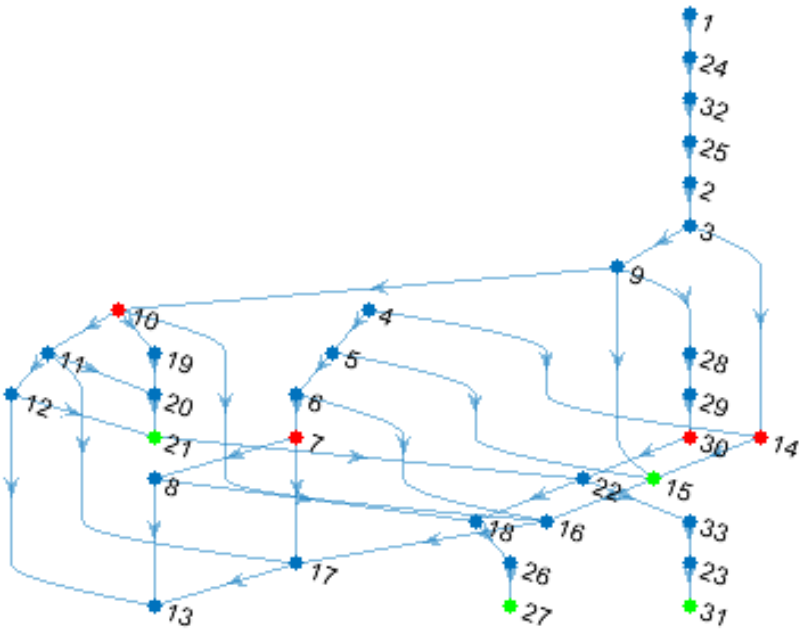

Figure 13: Example 3 of a WDN, where a red node is a faulty node and a green node an observer node

8 Conclusions

This thesis addresses the problem of sensor placement in water distribution networks. Sensors need to be placed to detect and isolate contamination faults. Based on previous literature [12], the problem of fault detection and isolation in water distribution networks was modeled with a linear, time-invariant, input-output state-space model and could also be represented using graph theory. A graph associated to a water distribution network is directed, weighted and may contain self-loops. In the literature [8] a graph topological condition for fault detection and isolation was stated for simple graphs. In order to use this graph topological condition it was shown that this sufficient condition is also applicable to a graph associated to a water distribution network. Using this graph topological condition an algorithm was developed. This algorithm computes where in the network observers need to be placed to guarantee fault detection and isolation under the assumption that it is known where the faults in the network could occur. Finally, some examples to illustrate the algorithm using Matlab were presented.

Future research could investigate sharper necessary conditions and a more efficient mathematical algorithm. Additionally, the Matlab code used for testing and illustrating the algorithm can be improved, as the currently used code is not able to redo step 1.

Bibliography

[1] Giuseppe Basile and Giovanni Marro. *Controlled and conditioned invariants in linear system theory*. Prentice Hall Englewood Cliffs, 1992.

[2] Demetrios G Eliades and Marios M Polycarpou. A fault diagnosis and security framework for water systems. *IEEE Transactions on Control Systems Technology*, 18(6):1254–1265, 2010.

[3] DG Eliades, TP Lambrou, CG Panayiotou, and MM Polycarpou. Contamination event detection in water distribution systems using a model-based approach. *Procedia Engineering*, 89:1089–1096, 2014.

[4] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18(3):636–653, 2010.

[5] Theofanis P Lambrou, Christos C Anastasiou, Christos G Panayiotou, and Marios M Polycarpou. A low-cost sensor network for real-time monitoring and contamination detection in drinking water distribution systems. *IEEE sensors journal*, 14(8):2765–2772, 2014.

[6] E Luvizotto Jr, MC Cavichia, P Vatavuk, and JGP Andrade. Nonmatrix gradient method for the simulation of water distribution networks. *Journal of Water Resources Planning and Management*, 139(4):433–439, 2012.

[7] RJ Patton, J Chen, and SB Nielsen. Model-based methods for fault diagnosis: some guide-lines. *Transactions of the Institute of Measurement and Control*, 17(2):73–83, 1995.

[8] Paolo Rapisarda, Anneroos RF Everts, and M Kanat Camlibel. Fault detection and isolation for systems defined over graphs. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 3816–3821. IEEE, 2015.

[9] S Rathi and R Gupta. Sensor placement methods for contamination detection in water distribution networks: A review. *Procedia Engineering*, 89:181–188, 2014.

[10] Takeaki Uno. A fast algorithm for enumerating bipartite perfect matchings. In *International Symposium on Algorithms and Computation*, pages 367–379. Springer, 2001.

[11] Peter van Thienen. Alternative strategies for optimal water quality sensor placement in drinking water distribution networks. 2014.

[12] Froukje Veldman-de Roo, Arturo Tejada, Henk van Waarde, and Harry L Trentelman. Towards observer-based fault detection and isolation for branched water distribution networks without cycles. In *Control Conference (ECC), 2015 European*, pages 3280–3285. IEEE, 2015.

Appendix

Matlab code

```
function [o] = sensorplacing(s,t,names,class)
%Sensorplacing: This algorithm determines which nodes in
%the WDN need to be assigned as observers to ensure fault
%detection and isolation, with known faulty nodes.

%Input for the algorithm is
%- An array of type double containing all the source nodes
%(s)
%- An array of type double containing all the target nodes
%(t)(in the same order as the corresponding source nodes)
%- An array of type cell stating the name of the nodes
%(Names)
%- An array of type string stating the class of each node
%(Class)(in the same order as the array containing the
% names of the nodes)

%Begin of algorithm
EdgeTable = table([s' t'], 'VariableNames',{'EndNodes'});
NodeTable = table(names, class, 'VariableNames',...
{'Name' 'Class'});
G = digraph(EdgeTable, NodeTable);

%Matlab has now generated a directed graph of the WDN using
%the input by making a table containing all the edges
%bewteen the source nodes (s)and the target nodes (t). A
%table containing information about the nodes,such as the
%name and class (which indicates whether or not a node is
%faulty, normal or not suited to be an observer) is also
%generated.From these tables a directed graph is made.

%step 1
faulty = G.Nodes.Name(G.Nodes.Class=='faulty');
%array containing all faulty nodes
faults = G.Nodes.Name(G.Nodes.Class=='faulty');
%array containing all faulty nodes,
%updates after every step
f = faults(1);
% fault to "fix"
po = G.Nodes.Name(G.Nodes.Class=='normal');
%array of nodes that could be observers
dfpo = distances(G,f,po);
```

```
%matrix displaying distances between nodes
dfpo(dfpo==inf) = 0;
%A distance of infinity means no path between the two nodes,
%set this to zero for the use of the max function
[M,I] = max(dfpo);
%M is the value of the largest path between the faulty node
%and possible observer, I the place in the array dfpo
obs = zeros(size(faulty,1),size(po,1));
pad = zeros(1,size(faulty,1));
pad(1) = M;
%distance between faulty node and chosen observer is stored
a = 1;
b = 0;

if M == 0
     text = 'No matching possible';
     fprintf(text)
end

%When no observers can be found that satifsy the
%constraints matlab prints text to tell us that.

obs(1,1) = str2double(po(I));
%which observer is chosen is stored
class(str2double(f)) = 'matched';
%class of fault is changed when a observer for this fault
%is found
class(str2double(po(I))) = 'observer';
%the class of the chosen observer is changed
NodeTable = table(names, class, 'VariableNames',...
{'Name' 'Class'});
%Table of nodes is updated
G = digraph(EdgeTable, NodeTable);
%graph is updated
faults = G.Nodes.Name(G.Nodes.Class=='faulty');
% array of faults is updated
i = 2;
%shows in which step the algorithm is

%The algorithm has to continue executing the steps of the
%while loop until all faults are "fixed", so for each
%fault the algorithm found an observer. Thus, step 2
%through k
    while str2double(faults)> 0
        if i == 1
            text = ['No constrained matching possible'...
```

```
            'with first observer at node %4d'];
            fprintf(text, obs(1,1))
            break
        end
%When step 2 is redone and there are no observers left to
%choose from Matlab informs us that it is not possible to
%find a constrained matching with the chosen first
%observer.

%From step 2 and onwards there are more constraints to be
%met than for step 1, the if-statement below ensures that
%in redoing a certain step all observers that were
%already "tried" in the step cannot be chosen again to
%avoid looping.

        f = faults(1);
        po = G.Nodes.Name(G.Nodes.Class=='normal');
        po = str2double(po);
        if b == 1
            for v = 1:a
                c = obs(i,v);
                po = po(po ~= c);
            end
        end
        m = G.Nodes.Name(G.Nodes.Class=='matched');
        o = G.Nodes.Name(G.Nodes.Class=='observer');
        dfo = distances(G,f,o);

%Due to the extra constraints it needs to be checked what
%the distances are between all already "matched" faults
%and all the assigned observers.
        for j = 1:size(m,1)
            dmpo = distances(G,m,po);
            dfpo = distances(G,f,po);
            po = po(dmpo(j,:) >= pad(j) & dfpo <= dfo(j));
%Nodes that satisfy the condition d(f1,o2)>=d(f1,o1) and
%d(f2,o1)>=d(f2,o2)meaning that the previous assigned
%observer stays in Wo and the to be assigned observer will
%also be part of Wo.
        end

%The if-statement below checks whether there are possible
%observers left to match the faulty node. If there are no
%possible observers left, then the fault is set to class
%"faulty" and all observers that could not be chosen in
%this step but could in general be an observer are set to
```

```
%class "normal". The last chosen observer in step i is set
%to "notnow". Subsequently, the nodetable, graph en faults
%are updated The algorithm skips the remaining of the while
%loop and starts the next iteration.
        if size(po,1) < 1
            i = i - 1;
            class(str2double(faulty(i))) = 'faulty';
            class(G.Nodes.Class=='notnow') = 'normal';
            ob = obs(i,:);
            ob = ob(ob>0);
            place = size(ob,2);
            class(ob(1,place)) = 'notnow';
            b = 1;
            NodeTable = table(names, class,...
            'VariableNames',{'Name' 'Class'});
            G = digraph(EdgeTable, NodeTable);
            faults = G.Nodes.Name(G.Nodes.Class=='faulty');
            a = a + 1;
            continue
        end

%This part ensures that an observer is choosen where there
%is a path from the fault to the observer and this path is
%the longest path from the fault to all possible observers.
        dfpo = distances(G,f,po);
        dfpo(dfpo==inf) = 0;
        [M,I] = max(dfpo);
        pad(i) = M;

%The if-statement below checks whether there is an observer
%left to match the faulty node. If there are no possible
%observers left, then the fault is set to class "faulty"
%and all observers that could not be chosenin this step but
%could in general be an observer are set to class "normal".
%The last chosen observer in step i is set to "notnow". And
%the nodetable,graph en faults are updated subsequently.
%The algorithm skips the remaining of the while loop and
%starts the next iteration.
        if M == 0
            i = i - 1;
            class(str2double(faulty(i))) = 'faulty';
            class(G.Nodes.Class=='notnow') = 'normal';
            ob = obs(i,:);
            ob = ob(ob>0);
            place = size(ob,2);
            class(ob(1,place)) = 'notnow';
```

```
            b = 1;
            NodeTable = table(names, class,...
            'VariableNames',{'Name' 'Class'});
            G = digraph(EdgeTable, NodeTable);
            faults = G.Nodes.Name(G.Nodes.Class=='faulty');
            a = a + 1;
            continue
        end

%When a possible observer satisfies all the constraints it
%is chosen as an observer. The fault is set to class
%"matched" and the graph is once again updated.
        class(po(I)) = 'observer';
        obs(i,a) = po(I);
        class(str2double(f)) = 'matched';
        NodeTable = table(names, class, 'VariableNames',...
        {'Name' 'Class'});
        G = digraph(EdgeTable, NodeTable);

%Making bipartite graph of all matched and observer nodes
%until now
        m = G.Nodes.Name(G.Nodes.Class=='matched');
        o = G.Nodes.Name(G.Nodes.Class=='observer');
        q = 1;
        r = size(m,1);
        source_nodes = cell(1,2*r);
        target_nodes = cell(1,2*r);

%The for-loop checks whether a faulty node and an observer
%node have a path that is in distance equal to the distance
%between the faulty nodes and its matched observer, only if
%this is true then both nodes are added to the list of
%target and source nodes to make the bipartite graph.
        for l = 1:r
            for n = 1:r
                dmo = distances(G,m(l),o(n));
                if dmo == pad(l)
                    source_nodes(q) = m(l);
                    target_nodes(q) = o(n);
                else
                    source_nodes(q) = {'0'};
                    target_nodes(q) = {'0'};
                end
                q = q+1;
            end
        end
```

```
%All source nodes and target nodes set to the type needed
%for creating a graph
        source_nodes = str2double(source_nodes);
        target_nodes = str2double(target_nodes);
        source_nodes = source_nodes(source_nodes>0);
        target_nodes = target_nodes(target_nodes>0);
        s_n = cellstr(string(source_nodes));
        t_n = cellstr(string(target_nodes));
        H = graph(s_n,t_n);

%Check whether H contains alternating cycles
%The bipartite graph is first split up in connected parts.
        bins = biconncomp(H,'OutputForm','cell');

        for e = 1:size(bins,2)
        w = bins(1,e);
        L = subgraph(H,w{1,1});

%if this is true then there is a cycle, then it is
%necessary to check wether the cycle is an alternating
%cycle. This is done by removing the edges present in
%the perfect matching we know, as the first fault is
%matched to the first observer etcetera. Matlab first
%establishes which edges need to be removed before
%removing them.
            if numedges(L) >= numnodes(L)
                q = 1;
                tn = zeros(1,r);
                for q = 1:r
                d = obs(q,:);
                tn(q) = d(find(d,1,'last'));
                end
                ks = L.Edges{:,1}(:,1);
                kt = L.Edges{:,1}(:,2);
                k = [ks;kt];
                K = [kt;ks];
                y = table(k,K);
                ks = m;
                kt = cellstr(string(tn'));
                k = [ks;kt];
                K = [kt;ks];
                Y = table(k,K);
                remove = setdiff(y,Y);
                so = remove{:,1};
                to = remove{:,2};
```

```
                Z = rmedge(L,so,to);
%After removing the edges present in the known perfect
%matching from part of the bipartite graph it is checked
%whether the new graph without theknown perfect matching
%contains enough edges for there to be another perfect
%matching.
%If this is the case the chosen observer is
%set to class "notnow" and the fault back to "faulty".
%Nodetable and graph are updated.
%If an alternating cycle is found there is no need to
%check the other parts of the bipartite graph.
                if 2*numedges(Z) == numedges(L)
                class(po(I)) = 'notnow';
                class(str2double(f)) = 'faulty';
                NodeTable = table(names, class,...
                'VariableNames',{'Name' 'Class'});
                G = digraph(EdgeTable, NodeTable);
              break
                end
            end
        end

%proceeding to next step of determining observers when
%the chosen observer in step i ensures a constrained
%matching.
        if class(str2double(f)) == 'matched'
            class(G.Nodes.Class=='notnow') = 'normal';
            i = i + 1;
            b = 0;
        end

%The array of the to be "matched" faults is updated.
        faults = G.Nodes.Name(G.Nodes.Class=='faulty');

    end

%Plots the constrained matching, where red nodes are
%faulty nodes and green nodes are the observers.
    g = plot(G);
    highlight(g,o,'NodeColor','g')
    highlight(g,m,'NodeColor','r')
 end
```