


Universitatea Politehnica Bucureşti
Facultatea de Automatică si Calculatoare
Departamentul de Automatică şi Ingineria Sistemelor

LUCRARE DE LICENŢĂ

Clasificare defecte într-o rețea de apă de mari dimensiuni

Absolvent
Cazan Cristian-Claudiu

Coordonator
Conf. dr. ing. Florin Stoican

Bucureşti, 2018

Cuprins

Listă de figuri

Listă de tabele

Listă de algoritmi

1. Introducere

1.1. Motivația alegerii temei

Transportul și distribuția apei reprezintă una dintre cele mai vechi preocupări inginerești de proporții, existând de mai mult de 4000 de ani. Civilizația minoică, localizată în insula Creta, este considerată a fi prima care a construit apeducte - structuri pentru transportul apei de la sursă către orașe - în 2500 î.Hr.

Deși majoritatea popoarelor din antichitate care s-au ocupat cu construcția apeductelor întrebuințau aceste sisteme pentru irigația pământului - ocupațiile de bază de atunci fiind în strânsă legătură cu agricultura - romanii au văzut în sistemele de provizionare a apei și un potențial imens în dezvoltarea civilizației, astfel ei sunt ei care aduc cele mai mari contribuții inginerești, apeductele construite de aceștia impresionând și astăzi prin grandoarea și iscusința cu care au fost construite.

Inovațiile în acest domeniu au suferit salturi bruște și puternice în momentul descoperirii unei noi relații matematice care transformă un parametru despre care se puteau face doar niște estimări grosiere într-o mărime bine definită și bine controlată. Istoric vorbind incipitul dezvoltării științei hidraulice s-a bazat pe relația descoperită de Arhimede din Siracuza în sec III î.Hr. $F = V_{obiect} * \rho_{lichid} * g$. O altă contribuție care are o deosebită importanță în domeniul tehnologiei de distribuiție a apei și nu numai o reprezintă tubul lui Pitot folosit la măsurarea vitezei fluidului, inventat de Henri Pitot în sec XVII. Din punct de vedere constructiv tubul are o formă de **L**, scufundarea acestuia într-un fluid (apă sau gaz) va determina creșterea nivelului și a presiunii până la o anumită limită ;Klopfenstein Jr , ecuația care guvernează depedența nivel - viteză este:

$$u = \sqrt{\frac{2(p_t - p_s)}{\rho}} \tag{1.1}$$

unde:

- u reprezintă viteza fluidului
- p_t reprezintă presiunea de stagnare
- p_s reprezintă presiunea statică
- ρ reprezintă densitatea fluidului

Mergând mai departe, alte contribuții importante apar din partea marilor matematicieni precum Daniel Bernoulli și Leonhard Euler, care au mărit spectrul mecanicii lui Newton și Leibniz spre aria hidraulicii și a termodinamicii. Fluidele considerate sunt incompresibile și au densitatea constantă în timp și uniform distribuită în spațiu. Bernoulli afirmă despre lichidele incompresibile că o creștere în viteză a lichidului este însoțită de o scădere a energiei potențiale a lichidului (i.e. presiune):

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = c \tag{1.2}$$

unde:

- v reprezintă viteza fluidului
- g reprezintă accelerația la care e supus fluidul
- z reprezintă elevația ștragulației conductei
- p reprezintă presiunea într-un anumit punct
- ρ reprezintă densitatea fluidului

În contextul în care se dorește analiza unui caz real este important ca toate diferențele între cazurile ideale și cazurile reale trebuie puse în evidență în mod matematic, astfel se particularizează ecuația generală Navier-Stokes pentru cazuri în care se cunosc anumiți parametrii ai sistemului de analizat. Spre exemplu ecuația Poisuille care modelează începutul fluxului de apă într-o conductă este ;Elger şi Roberson :

$$\frac{\partial u}{\partial t} = \frac{G}{\rho} + \nu(\frac{\partial^2 u}{\partial t^2} + \frac{1}{r}\frac{\partial u}{\partial r}) \tag{1.3}$$

unde:

- u reprezintă viteza lichidului prin conductă
- t reprezintă timpul
- G reprezintă diferența de presiune
- ρ reprezintă densitatea lichidului
- ν reprezintă vâscozitatea cinematică
- r reprezintă poziția

Se poate observa că pe măsură ce modelul matematic se apropie de realitate, complexitatea acestuia crește și pentru fiecare situație specială - spre exemplu analiza presiunii la introducerea apei într-o conductă vs. analiza presiunii când conducta este încărcată cu apă - are nevoie de o ecuație specială sau de o particularizare a ecuației Navier-Stokes, pentru care încă nu se conoaște dacă există soluții pentru cazul cu 3 dimensiuni și dacă soluțiile acestea sunt netede.

Ținând cont de importanța apei în desfășurarea activităților cotidiene atât pentru oameni cât și pentru actorii importanți ai industriei, este o condiție sine-qua-non ca un oraș să aibă un sistem performant și rezistent la defecte pentru distribuția apei. În contextul actual al dezvoltării tehnologiei este natural să folosim tehnici moderne de monitorizare a diferiților parametrii din cadrul unei rețele pentru a putea face o analiză riguroasă și eficientă cu referire nu numai la mentenanță ci și la consumul global și local în ideea îmbunătățirii și reducerii pierderilor.

1.2. Expunerea problemei

În această lucrare se va aborda problematica identificării prezenței unui defect - *Fault detection* și izolarea defectului *Fault isolation* într-o regiune a rețelei.

O rețea de apă poate fi privită ca un graf neorientat $G = (V, E)$ unde V este mulțimea nodurilor rețelei - acestea reprezentând o abstractizare asupra componentelor precum:

- rezervoare

- tancuri de apă
- puncte de distribuție

E este mulțimea muchiilor reprezentând de fapt țevile care fac legătura între noduri.

Mergând mai departe cu abstractizarea se pot considera rețele de apă active și rețele de apă pasive. Diferența între cele două făcându-se în baza pompelor de apă amplasate în zonele unde presiunea sau elevația vin în detrimentul distribuției apei.

Rețelele de apă care vor fi tratate în această lucrare fac parte din categoria pasivă, astfel putem diviza mulțimea nodurilor V în V^t și în V^j reprezentând mulțimea nodurilor de tip tanc și mulțimea nodurilor joncțiune, cu proprietatea că $V = V^t \cup V^j$. Tancurile și rezervoarele dintr-o rețea de apă au proprietatea că nivelul de apă din acestea se va menține la un nivel oarecum staționar, astfel simulările din capitolele viitoare se vor axa pe nodurile simple de tip joncțiune, deci mulțimea de interes în acest caz va fi V^j pentru care cunoaștem cardinalul.

Caracteristicile care se pot recolta dintr-o rețea de apă pot varia în funcție de elementul inspectat și de senzorii dispuși în rețea, astfel pentru fiecare nod $n_i \in V^j$ putem defini la fiecare moment de timp

- presiunea $p_i(t)$ - măsurată în metri coloană de apă $mH2O$, mărime influențată puternic de presiunea interioară a nodului și de eventualele perturbații exterioare i.e. scurgeri de apă prin țevi
- 'cererea' $d_i(t)$ - măsurată L/s, mărime ce caracterizează profilul de utilizare al utilizatorilor de-a lungul unei zile i.e. debitul de apă care ajunge la consumatori. Acest debit poate varia de-a lungul zilei, putem distinge de exemplu intervale de timp în care cererea este foarte mică și rețeaua intră în regim staționar

De asemenea pentru fiecare conductă a rețelei $e_{ij} \in E$ putem măsura viteza lichidului $v_{ij}(t)$.

Pentru a putea rezolva problema de *Fault Detection and Isolation* este importantă găsirea unei modalități eficente de selecție și prelucrare a datelor de la rețea.Mai mult, punând în lumină aspectul ingineresc al problemei, trebuie găsită o submulțime $V_{opt} \subset V^j$ ai cărei elemente pot aduce informații necesare și suficiente pentru a detecta un defect într-o acoperire destul de mare a rețelei.

1.3. Exemplul de lucru

În următoarele capitole și în implementarea lucrării consider rețeaua din Hanoi iar pentru simularea scenariilor propuse voi folosi biblioteca și suita de funcții **EPANET** - Environmental Protection Agency NETwork

Rețeaua Hanoi constă într-o mulțime de noduri de tip joncțiune V^j cu $|V^j| = 31$ și mulțimea V^t cu $|V^t| = 1$, ilustrată în figura de mai jos:

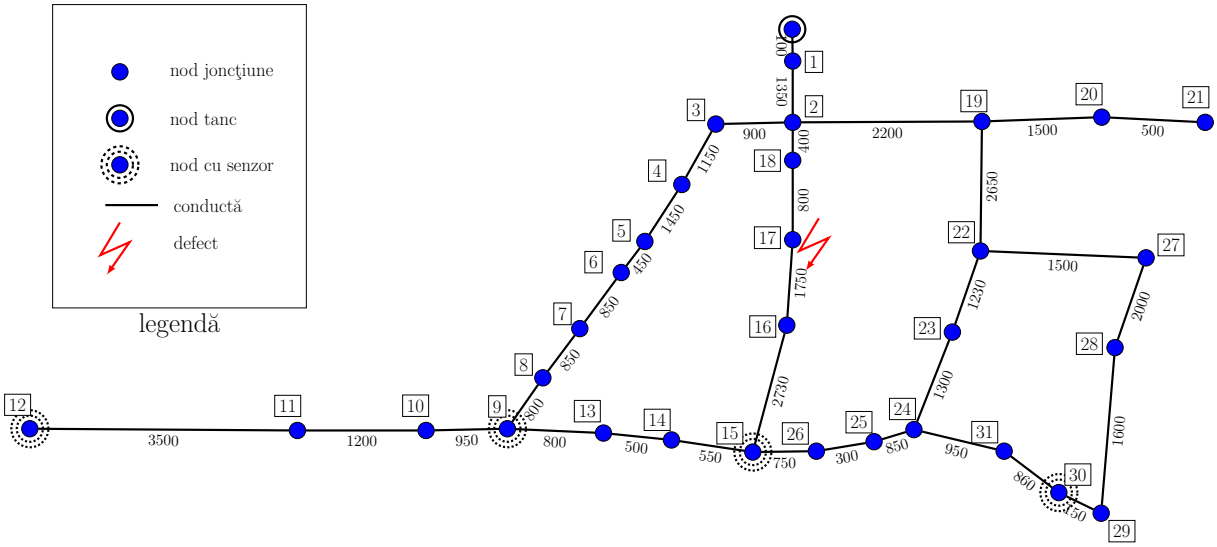

Figura 1.1.: Graful rețelei de apă din Hanoi

După cum se poate observa în figura de mai sus au fost reprezentate două tipuri de node pasive, anume tancurile și joncțiunile. În cazul apariției unui defect în rețea, este important de luat în considerație modalitatea în care acesta va influența dinamica rețelei, spre exemplu este de la sine înțeles că dacă se cosideră un defect în nodul cu indicele 17 - i.e. în acest nod au apărut anumite scurgeri care afectează fluxul de apă către consumatori - nodurile în care se va observa o modificare puternică a caracteristicilor (presiune și debit) vor face parte din mulțimea nodurilor adiacente rețelei $S = V_{16}, V_{18}$, deși pare o concluzie naturală, o modelare matematică riguroasă din care să se tragă această nu este o problemă foarte ușor de rezolvat, anumiți parametrii fiind extrem de greu de estimat chiar și în cazul în care se consideră un regim staționar.

2. Simulări și software folosit

2.1. Dificultatea estimărilor parametrilor într-o rețea de apă

Găsirea unui set de ecuații al cărei soluție să conducă la o estimare îndeajuns de bună pentru control este o condiție sine qua non pentru detecția unui defect și izolarea acestuia în cadrul nodurilor rețelei. Astfel după cum a fost expus în capitolul 1 ecuațiile care guvernează relațiile intre viteza prin conducte și presiune dintr-un anumit punct sunt particularizări ale ecuațiilor Bernoulli-Euler sau Navier-Stokes. În cadrul unei rețele de apă a unui oraș, complexitatea rezolvării problemei crește semnificativ din varii motive precum:

- ansamblul de coduncte și noduri interconectate dă naștere unui sistem fizic greu de modelat matematic
- parametrii care pot influența calitatea soluțiilor precum: tipul materialului conductei și al nodului, elevația fiecărui nod, rugozitatea fiecărei conducte și depunerile de pe aceasta
- apariția unor factori exogeni care pot fi uneori greu de estimat - tiparul de utilizare al rețelei de către consumatori poate varia puternic
- apariția defectelor precum scurgerile în proximitatea unui nod

Ținând cont de complexitatea problemei în regim dinamic pentru a putea obține o soluție de regim staționar a rețelei este necesar să ignorăm evenimentele imprevizibile precum apariția unei scurgeri sau variațiile bruște ale consumului.

Ecuațiile de regim staționar includ condiții de conservare fluxului de apă:

$$\sum_{j=1}^{n} \mathbf{B}_{ij}\mathbf{q}_j = \mathbf{d}_i \tag{2.1}$$

Unde q_i reprezintă debitul prin fiecare conductă iar $\mathbf{B}$ reprezintă matricea de adiacență a rețelei la echilibru, definită astfel

$$\mathbf{B}_{ij} = \begin{cases} 1, & \text{conducta j intră în nodul i} \\ 0, & \text{conducta j nu este conectată la nodul i} \\ -1, & \text{conducta j iese din nodul i} \end{cases} \tag{2.2}$$

Partea de estimare a diferenței de presiuni (în engl. "Head-Flow differential") între două noduri interconectate se face utilizând formula Hazen-Williams ;Sanz Estapé :

$$\mathbf{h}_i - \mathbf{h}_j = \frac{10.67 \cdot L_\ell}{C_\ell^{1.852} \cdot D_\ell^{4.87}} \cdot \mathbf{q}_\ell \cdot |\mathbf{q}_\ell|^{0.852} \tag{2.3}$$

unde:

- **h** reprezintă presiunea - măsurată de obicei în metru coloană de apă
- C_l reprezintă coeficientul de rugozitate al conductei
- D_l reprezintă diametrul conductei
- L_l reprezintă lungimea conductei
- q_l reprezintă debitul

Din ecuația empirică (2.1) termenul $R_{ij} = \frac{10.67 \cdot L_\ell}{C_\ell^{1.852} \cdot D_\ell^{4.87}}$ reprezintă rezistența conductei ij iar dual, putem obține conductivitatea conductei $G_{ij} = \frac{1}{R_{ij}}$

Având la dispoziție (2.1) și (2.3) putem exprima dependența debit presiune în regim staționar sub o formă matriceală compactă și cu o structură neliniară:

$$\mathbf{BG}\left[\left(-\mathbf{B}^\top \mathbf{h} + \mathbf{B}_f^\top \mathbf{h}_f\right) \times \left|-\mathbf{B}^\top \mathbf{h} + \mathbf{B}_f^\top \mathbf{h}_f\right|^{-0.46}\right] = \mathbf{d} \tag{2.4}$$

unde s-au luat în calcul și nodurile care au variații de presiune foarte mici - spre exemplu nodurile de tip tanc și nodurile de tip rezervor - termenul $\mathbf{B}_f^\top \mathbf{h}_f$ reprezintă contribuția acestor noduri la starea de echilibru a rețelei.

Din cauza dificultății rezolvării unei ecuații matriceale neliniare, software-ul specializat trebuie să folosească diferite metode de optimizare (”Solver”) pentru a putea obține o diferență cât mai mică între cazul estimat și rezultatul real al ecuației. Este important de reținut faptul că rezolvarea problemelor de programare neliniară cu constrângeri poate generea de fapt o problemă NP-completă, sau în unele cazuri chiar NP-dură ;Karp .

2.2. Simulări folosind biblioteca EPANET

Dezvoltat la începutul anilor 90’ de către USEPA (United States Environmental Protection Agency), EPANET a fost inițial privit ca un instrument pentru cercetare, acesta a devenit un standard de industrie la capitolul simulărilor software robuste pentru rețele de apă, foarte multe pachete software proprietare de simulare hidraulică se bazează masiv pe EPANET, diferențele apărând la design-ul interfeței grafice și a manipulării datelor. Aceast program de simulare oferă utilizatorului posibilitatea de a-și defini într-un mod interactiv o rețea de apă configurând tipul de nod, legătura între oricare două noduri și posibilitatea de a adăuga și elemente active în rețea, pompe. Pentru a simula utilizatorul trebuie să își definească pentru fiecare nod un anumit debit cerut de utilizatori, o elevație, și o legătură cu alte noduri. Simularea se va desfășura pe o perioadă de timp definită cu pasul de eșantionare cât mai convenabil ;Rossman .

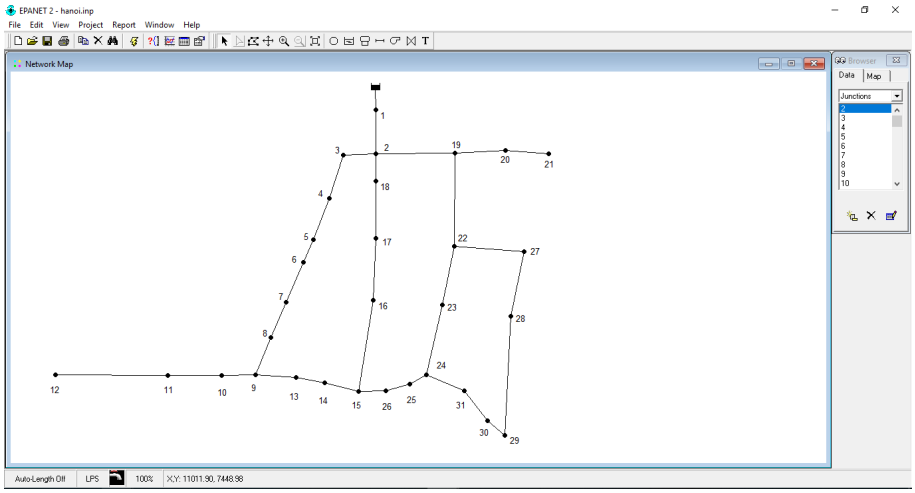

Figura 2.1.: Simulatorul EPANET

În urma execuției simulării EPANET va stoca toate datele în memorie și va putea reliza grafice și alte interogări complexe.

Modalitatea în care simulatorul EPANET reușește să obțină datele de simulare este prin implementarea eficientă a ecuațiilor Hazen-Williams (2.1) Darcy-Weisbach și Chezy-Manning, la fiecare perioadă de eșantionare algoritmul bazat de metoda gradientului rezolvă ecuațiile matriceale neliniare.

3. Detecția și izolarea defectelor

3.1. Testing Decection

Anexe

A. Fișiere sursă

```
# epanet toolkit
import json

import numpy as np
import pandas as pd
from epanettools.epanet2 import *
from epanettools.epanettools import *
from plotly import tools
from plotly.graph_objs import *
# plotting imports
from plotly.offline import plot

# extending the EPANetSimulation class to ease acces to
# simulation routines
class ENSim(EPANetSimulation):
    EN_INIT = 10

    def __init__(self, network_file, pdd=False):
        # careful when using pdd = true for residues simulations
        self.json_sim = {}
        super().__init__(network_file, pdd)

    def set_emitter(self, node_index, emitter_val):
        if self.network.nodes[node_index].node_type is EN_JUNCTION:
            ENSim._getncheck(self.ENsetnodevalue(node_index, EN_EMITTER, emitter_val
    ))

    def set_basedemand(self, node_index, demand_val):
        if self.network.nodes[node_index].node_type is EN_JUNCTION:
            ENSim._getncheck(self.ENsetnodevalue(node_index, EN_BASEDEMAND,
    demand_val))

    def set_emitters(self, emitter_info=None):

        if emitter_info is None:
            # if arg is none reset emitter values
            for node_index in self.network.nodes:
                self.set_emitter(node_index, 0)
        else:

            for node_index, emitter_val in emitter_info:
                self.set_emitter(node_index, emitter_val)

    def get_nodes_data(self, data_query, emitter=(1, 0)):

        no_nodes = ENSim._getncheck(self.ENgetcount(EN_NODECOUNT)) - ENSim.
    _getncheck(self.ENgetcount(EN_TANKCOUNT))
        t_step = 1
        node_values = {}

        for queries in data_query:
            node_values[queries] = [[] for _ in range(no_nodes)]

        node_values["EMITTER_NODE"] = emitter[0]
        node_values["EMITTER_VAL"] = emitter[1]

        # initialize network for hydraulic process
```

```
55
        ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))
57
        while t_step > 0:
59
            self.ENrunH()
61
            for node_index in range(1, no_nodes + 1):
63              for query_type in data_query:
                    ret_val = ENSim._getncheck(self.ENgetnodevalue(node_index, eval(
    query_type)))
65                  node_values[query_type][node_index - 1].append(ret_val)
67          t_step = ENSim._getncheck(self.ENnextH())
69      for key in node_values:
            node_values[key] = np.transpose(node_values[key]).tolist()
71
        return node_values
73
    def get_links_data(self, data_query, emitter=(1, 0)):
75
        no_links = self.ENgetcount(EN_LINKCOUNT)[1]
77      t_step = 1
        link_values = {}
79
        for queries in data_query:
81          link_values[queries] = [[] for _ in range(no_links)]
83      link_values["EMITTER_NODE"] = emitter[0]
        link_values["EMITTER_VAL"] = emitter[1]
85
        # initialize network for hydraulic process
87
        ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))
89
        while t_step > 0:
91          ENSim._getncheck(self.ENrunH())
93          for link_index in range(1, no_links + 1):
                for query_type in data_query:
95                  ret_val = ENSim._getncheck(self.ENgetlinkvalue(link_index, eval(
    query_type)))
                    link_values[query_type][link_index - 1].append(ret_val)
97
            t_step = ENSim._getncheck(self.ENnextH())
99
        for key in link_values:
101         link_values[key] = np.transpose(link_values[key]).tolist()
103     return link_values
105 def query_network(self, sim_dict):
        """
107     :param sim_dict: a dict containing info about the network
        has the form
109     {
            simulation_name : "name",
111         simulation_type: "H" or "Q"
            emitter_values : [ (node_index, emitter_value) ]
113         query : {
                nodes : [ "EN_PRESSURE"
115             ]
                links : [ "EN_VELOCITY"
117             ]
            }
```

```
119         }
            :return: JSON with required data
121         output_json format:
            {
123             SIM_NAME = simulation-name
                NODE_VALUES = [
125                 {
                        "EN_PRESSURE" : [ [values for each node]]
127                     "EMITTER_VAL" :
                        "EMITTER_NODE" :
129                 }
                ]
131
            }
133
            """
135
            # for the moment i'll treat only hydraulic simulations :)
137
            # initialize network simulaton
139
            ENSim._getncheck(self.ENopenH())
141
            # initialize session
143         ENSim._getncheck(self.ENinitH(ENSim.EN_INIT))

145         node_query = False
            link_query = False
147         simulations = False

149         # check json for querried data

151         # node info:
            try:
153             if sim_dict["query"]["nodes"]:
                    node_query = True
155         except KeyError:
                node_query = False
157
            # link info
159         try:
                if sim_dict["query"]["links"]:
161                 link_query = True
            except KeyError:
163             link_query = False

165         # emitter info:
            try:
167             simulations = sim_dict["emitter_values"]
            except KeyError:
169             simulations = False

171         if simulations:
                node_values = []
173             link_values = []

175             # in order to plot residues we need to simulate the case where there is
    no leakage

177             self.set_emitters()
                if node_query:
179                 node_values.append(
                        self.get_nodes_data(sim_dict["query"]["nodes"]))
181
                if link_query:
183                 link_values.append(
```

```
                    self.get_links_data(sim_dict["query"]["links"]))

            for node_index, emitter_value in simulations:
                print("Simulating emitter in node no {} with value {}".format(
    node_index, emitter_value))

                self.set_emitter(node_index, emitter_value)

                if node_query:
                    node_values.append(
                        self.get_nodes_data(sim_dict["query"]["nodes"], emitter=(
    node_index, emitter_value)))

                if link_query:
                    link_values.append(
                        self.get_links_data(sim_dict["query"]["links"], emitter=(
    node_index, emitter_value)))

                # reset emitter values everywhere in network
                self.set_emitters()

        else:

            if node_query:
                node_values = [self.get_nodes_data(sim_dict["query"]["nodes"])]
            else:
                node_values = []

            if link_query:
                link_values = [self.get_links_data(sim_dict["query"]["links"])]
            else:
                link_values = []

        ENSim._getncheck(self.ENcloseH())
        ENSim._getncheck(self.ENclose())

        self.__init__(self.OriginalInputFileName)
        self.json_sim = {
            "SIM_NAME": sim_dict["simulation_name"],
            "NODE_VALUES": node_values,
            "LINK_VALUES": link_values
        }
        return self.json_sim

    def get_time_step(self, pattern_id=1):
        """
        returns the time_step of the network in minutes
        :param pattern_id:
        :return:
        """
        return (24 * 60) / ENSim._getncheck(self.ENgetpatternlen(pattern_id))

    def plot(self, json_data, residues=False):
        """
        utility function used to plot data from network simulations
        WIP
        :param json_data:
        :return:
        """
        values = json_data["NODE_VALUES"]
        date_range = pd.date_range('1/1/2018', periods=97, freq='15min')

        if residues:
            # consider the first value of the JSON as the reference
            ref = values[0]
            for emitter in values:
```

```
247                 trace = []
                    data = np.transpose(emitter["EN_PRESSURE"]) - np.transpose(ref["
    EN_PRESSURE"])
249
251                 for node_index, vals in enumerate(data):
                        trace.append(Scatter(
253                         x=date_range,
                            y=vals,
255                         name="node{}".format(node_index+1)
                        )
257                     )
                    layout = dict(
259                     title = "Ressidues with emitter in node {}, val = {}".format(
    emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
                    )
261
                    fig = dict(data=trace, layout=layout)
263                 plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
    ], emitter["EMITTER_VAL"]))

265         else:
                for emitter in values:
267                 trace = []
                    data = np.transpose(emitter["EN_PRESSURE"])
269                 for node_index, vals in enumerate(data):
                        trace.append(Scatter(
271                         x=date_range,
                            y=vals,
273                         name="node{}".format(node_index+1)
                        )
275                     )
                    layout = dict(
277                     title = "Pressure with emitter in node {}, val = {}".format(
    emitter["EMITTER_NODE"], emitter["EMITTER_VAL"])
                    )
279
                    fig = dict(data=trace, layout=layout)
281                 plot(fig, filename= "Plot_node{}val{}".format(emitter["EMITTER_NODE"
    ], emitter["EMITTER_VAL"]))

283     def save_data(self, path=None):

285         try:
                with open(path, "wt") as file:
287                 file.write(json.dumps(self.json_sim))
            except IOError:
289             print("Could not write to file {}".format(path))

291     @staticmethod
        def write_json(output_json, path):
293
            json_data = json.dumps(output_json)
295         with open(path, "wt") as f:
                f.write(json_data)
297
        @staticmethod
299     def _getncheck(ret_val):

301         # check the return code
            if isinstance(ret_val, list):
303             if ret_val[0] == 0:
                    # everything OK
305                 return ret_val[1]
                else:
307                 err_msg = ENgeterror(ret_val[0], 100)
```

```
                raise EpanetError(err_msg)
309         else:
                if ret_val is not 0:
311                 err_msg = ENgeterror(ret_val, 100)
                    raise EpanetError(err_msg)
313
315 def en_check(func):
        def func_wrapper(*args):
317         ret_val = func(args)

319         # check the return code
            if isinstance(ret_val, list):
321             if ret_val[0] == 0:
                    # everything OK
323                 return ret_val[1]
                else:
325                 err_msg = ENgeterror(ret_val[0], 100)
                    raise EpanetError(err_msg)
327         else:
                if ret_val is not 0:
329                 err_msg = ENgeterror(ret_val, 100)
                    raise EpanetError(err_msg)
331

333 class EpanetError(Exception):

335     def __init__(self, err_msg):
            super().__init__(err_msg)
337

339 def run_simulation(network, pdd, query_dict):

341     es = EPANetSimulation(network, pdd)

343     print("Running {}".format(query_dict["simulation_name"]))
        ret_vals = []
345
        print(query_dict)
347     for emitter, emitter_val in query_dict["emitter_values"]:
            print("for node {} simulating emitter_Val {}".format(emitter, emitter_val))
349
            # modify current network and and save inp temp file
351
            es.ENsetnodevalue(emitter, EN_EMITTER, emitter_val)
353
            es.ENsaveinpfile("temp.inp")
355         print(es.ENsetnodevalue(emitter, EN_EMITTER, 0))

357         e2 = EPANetSimulation("temp.inp", pdd)
            e2.ENsetnodevalue(emitter,EN_EMITTER,emitter_val)
359
            e2.run()
361
            node_vals = {}
363         link_vals = {}
            for node_query in query_dict["query"]["nodes"]:
365             node_vals[node_query] = []

367         for link_query in query_dict["query"]["links"]:
                link_vals[link_query] = []
369
            for node_query in query_dict["query"]["nodes"]:
371             for node in e2.network.nodes:
                    node_vals[node_query].append(e2.network.nodes[node].results[eval(
    node_query)])
```

```
373
            for link_query in query_dict["query"]["links"]:
375                 for link in e2.network.links:
                        link_vals[link_query].append(e2.network.links[link].results[eval(
    link_query)])
377
379             ret_vals.append({
                    "EMITTER_VAL" : emitter_val,
381                 "EMITTER_NODE" : emitter,
                    "NODE_VALS" : np.transpose(node_vals).tolist(),
383                 "LINK_VALS" : np.transpose(link_vals).tolist()
                })
385         return ret_vals

387 #TODO metoda pentru modificarea demand-ului pe nod!!!
    if __name__ == '__main__':
389     es = ENSim("data/hanoi.inp", pdd=False)

391     test_vals = [val for val in range(32) if val % 2 == 0]
        train_vals = [val for val in range(32) if val % 2 == 1]
393

395     intense_leak = [35, 40, 50, 60, 100]

397     nodes = list(range(1, 32))

399     emitter_test = [ (node, val) for node in nodes for val in test_vals]
        emitter_train = [(node, val) for node in nodes for val in train_vals]
401     emitter_intense_leak = [(node, val) for node in nodes for val in intense_leak]

403     train_dataset = {
            "simulation_name": "Hanoi train simulation",
405         "simulation_type": "H",
            "emitter_values": emitter_train,
407         "query": {

409             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
                "links": ["EN_VELOCITY"]
411         }

413     }

415     test_dataset = {
            "simulation_name": "Hanoi test simulation",
417         "simulation_type": "H",
            "emitter_values": emitter_test,
419         "query": {

421             "nodes": ["EN_PRESSURE", "EN_DEMAND"],
                "links": ["EN_VELOCITY"]
423         }

425     }

427     test2_dataset = {

429         "simulation_name": "Hanoi intense leak simulation",
            "simulation_type": "H",
431         "emitter_values": emitter_intense_leak,
            "query": {
433
                "nodes": ["EN_PRESSURE", "EN_DEMAND"],
435             "links": ["EN_VELOCITY"]
            }
437
```

```
    }

    data_train = es.query_network(train_dataset)
    es.save_data("train_set.json")
    data_test = es.query_network(test_dataset)
    es.save_data("test_set.json")
    data_test2 = es.query_network(test2_dataset)
    es.save_data("test2_set.json")
```

Listing A.1: Wrapper EPANET – fișier complet

Bibliografie

Elger, Donald F şi Roberson, John A (). *Engineering fluid mechanics*. Wiley Hoboken (NJ).

Karp, Richard M (). *On the computational complexity of combinatorial problems*. Networks, vol. 5(1), pags. 45–68.

Klopfenstein Jr, Rex (). *Air velocity and flow measurement using a Pitot tube*. ISA transactions, vol. 37(4), pags. 257–263.

Rossman, Lewis A şi alţii (). *EPANET 2: users manual*.

Sanz Estapé, Gerard (). *Demand modeling for water networks calibration and leak localization*.