POLITECHNIKA POZNAŃSKA
WYDZIAŁ AUTOMATYKI
ROBOTYKI I ELEKTROTECHNIKI

ROZPRAWA DOKTORSKA

ALGORYTMY WSPOMAGANIA DECYZJI W ZAKRESIE DZIAŁAŃ NAPRAWCZYCH NA PRZYKŁADZIE SYSTEMU DYSTRYBUCJI WODY

mgr inż. Ariel ANTONOWICZ

Promotor
prof. dr hab. inż. Andrzej URBANIAK
Politechnika Poznańska

Pragnę podziękować wszystkim,
którzy przyczynili się do powstania niniejszej pracy doktorskiej.

Promotorowi, Panu profesorowi dr hab. inż. Andrzejowi Urbaniakowi,
za wyrozumiałość, cierpliwość, zaangażowanie oraz cenne uwagi i sugestie.

Kochanej żonie Karolinie za możliwość realizowania się i spełniania marzeń.

Rodzicom za trud wychowania, wsparcie i inspirację do działania.

Teściowej za wrażliwość, wiarę we mnie i nieocenioną pomoc.

Kierownictwu oraz pracownikom Miejskiego Przedsiębiorstwa Energetyki Cieplnej,
Wodociągów i Kanalizacji. Sp. z o.o. w Środzie Wielkopolskiej za możliwość współpracy.

Streszczenie

Praca doktorska pod tytułem „*Algorytmy wspomagania decyzji w zakresie działań naprawczych na przykładzie systemu dystrybucji wody*" jest przykładem synergii trzech dyscyplin naukowych takich jak: automatyka, elektronika i elektrotechnika, informatyka techniczna i telekomunikacja oraz inżynieria środowiska, górnictwo i energetyka. Celem niniejszej rozprawy jest weryfikacja tezy, iż dla zdefiniowanego problemu decyzyjnego możliwe jest określenie kryteriów wyboru oraz ograniczeń, które pozwolą zachować największą sprawność systemu wodociągowego w sytuacji wystąpienia wielu awarii. Na podstawie modelu hydraulicznego sieci wodociągowej, informacji o stanie urządzeń, materiałów eksploatacyjnych, danych związanych z liczbą dostępnych ekip naprawczych, informacji o alokacji zasuw oraz lokalizacji awarii opracowane rozwiązanie pozwoli ustalić kolejność ich naprawy, osiągając zamierzoną funkcję celu (minimalizacja czasu naprawy w celu jak najszybszego przywrócenia ciągłości procesu dystrybucji wody). Zaproponowane rozwiązanie stanowi solidny fundament dla Systemów Wspomagania Decyzji oraz Systemów Ekspertowych, ponieważ opracowane jest w oparciu o otwarte oprogramowanie. Oznacza to, że na podstawie tzw. licencji wolnego oprogramowania każdy może analizować, zmieniać i rozpowszechniać zmodyfikowaną wersję.

Rozwiązanie zawiera 7 algorytmów, których połączenie umożliwi realizację ww. celu. Zadaniem pierwszego zaproponowanego algorytmu jest sklasyfikowanie poszczególnych elementów sieci wodociągowej. Ma to na celu sprawdzenie krytyczności i określenia współczynnika ważności elementu na tle całego zbiorowego systemu zaopatrzenia w wodę. Drugi algorytm na podstawie danych o topologii sieci oraz informacji o identyfikatorze analizowanego węzła wyznacza tzw. ślad wody. W oparciu o lokalizację infrastruktury krytycznej określana jest droga od źródła wody (stacji uzdatniania wody lub zbiornika) do węzła, który reprezentuje analizowanego odbiorcę. Kolejny algorytm na podstawie danych o lokalizacji zasuw odcinających odpowiada za wyznaczenie segmentów występujących na sieci wodociągowej oraz zbioru zasuw niezbędnych do jego izolacji. Algorytm czwarty odpowiada za klasyfikację zgłoszonych awarii. Na podstawie określonych współczynników awaria zostaje sklasyfikowana do jednej z czterech dostępnych klas. Algorytm piąty na podstawie wyników działania poprzednich algorytmów nadaje poszczególnym awariom (o ile to konieczne) priorytet,

który brany jest pod uwagę w sytuacji szeregowania zleceń naprawczych. Zadaniem szóstego algorytmu jest określenie możliwych połączeń awarii w jedno zlecenie naprawcze. Na podstawie wiedzy o lokalizacji awarii oraz ich typu istnieje możliwość agregacji awarii w celu minimalizacji sumarycznej liczby zasuw niezbędnych do zamknięcia. Ostatni algorytm odpowiada za szeregowanie zadań. W oparciu o listę awarii, funkcję celu i ograniczenia, określona zostaje marszruta dla każdej ekipy naprawczej oraz lista zawierająca propozycję kolejności napraw awarii.

Dysertacja składa się z dziewięciu rozdziałów, spisu treści, wykazu literatury, spisu rysunków, spisu tabel, wykazu akronimów i ważniejszych oznaczeń oraz załączników. Każdy rozdział rozpoczyna się krótkim wprowadzeniem przedstawiającym omawiane w nim zagadnienie.

W rozdziale pierwszym (wprowadzeniu) przedstawiona została motywacja podjętego tematu pracy. Następnie określony został przedmiot pracy (problem badawczy), cel pracy oraz zawartość rozprawy doktorskiej.

W rozdziale drugim omówione zostały wybrane nowoczesne systemy sterowania i monitorowania. Przedstawiono ogólną i warstwową strukturę sterowania oraz opisano metody zarządzania systemami informatycznymi w procesie eksploatacji z wykorzystaniem systemów SCADA, Systemów Wspomagania Decyzji oraz Systemów Ekspertowych. Ponadto rozdział ten porusza tematykę integralności (interoperacyjności) złożonych systemów informatycznych.

Rozdział trzeci zawiera wstęp teoretyczny, wprowadzający w tematykę systemów zbiorowego zaopatrzenia w wodę. W rozdziale tym zawarto podstawowe informację o procesie uzdatniania wody, budowie i rodzajach sieci wodociągowych oraz pokrótce opisano problem analizy krytyczności elementów wchodzących w skład takich sieci.

Rozdział czwarty poświęcony został metodyce modelowania procesu dystrybucji wody z wykorzystaniem narzędzi takich jak EPANET czy WNTR. W rozdziale zaprezentowano przykład modelowania testowego modelu sieci wodociągowej, poruszono tematykę kalibracji modeli oraz przedstawiono dwa sposoby modelowania awarii.

Rozdział piąty poświęcony został przeglądowi literatury, na temat rozwiązań informatycznych, które w sposób szczególny mogą przyczyniać się do usprawnienia procesu typowania zasuw niezbędnych do odseparowania segmentu, w którym wystąpiła jedna lub kilka awarii. W rozdziale przedstawiono również wybraną metodę typowania zasuw do zamknięcia wraz z jej niewielką modyfikacją.

W rozdziale szóstym poruszono problematykę podejmowania decyzji o kolejności naprawiania awarii w systemach wodociągowych. Wskazano cechy awarii, które należy rozpatrzeć w procesie ich klasyfikacji oraz omówiono proces priorytetyzacji. Ponadto rozdział ten poświęcony został charakterystyce problematyki szeregowania zadań z punktu widzenia zarządzania ekipami naprawczymi. Przedstawiono metody rozwiązywania problemów szeregowania zadań oraz wykorzystanie algorytmów sztucznej inteligencji, które w sposób szczególny mogą przyczyniać się do usprawnienia procesu szeregowania zadań.

W rozdziale siódmym przedstawiono grupę algorytmów stanowiących fundament systemu wspomagania decyzji, których zadaniem jest poprawa efektywności procesu przywracania ciągłości wody w sytuacji po katastroficznej. Algorytmy zostały opracowane na podstawie analizy doświadczeń eksperta dziedzinowego z *Miejskiego Przedsiębiorstwa Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej* oraz zbioru artykułów naukowych.

Rozdział ósmy zawiera wyniki badań symulacyjnych dotyczących zaproponowanej metodyki poprawy efektywności procesu przywracania ciągłości dystrybucji wody w sytuacji po katastroficznej. W pierwszej części rozdziału omówiono zrealizowane środowisko symulacyjne oraz przedstawiono przygotowane modele hydrauliczne. W dalszej części przedstawiono wygenerowane scenariusze testowe oraz przedstawiono przyjęte założenia. Rozdział zamyka wyniki analizy działania i poprawności zaproponowanego systemu.

W rozdziale dziewiątym zamieszczono podsumowanie oraz wnioski końcowe rozprawy.

Abstract

The doctoral thesis entitled "Decision support algorithms in the field of repair actions on the example of the water distribution system" is an example of the synergy of three scientific disciplines such as: automation, electronics and electrical engineering, technical information technology and telecommunications as well as environmental engineering, mining and energy. The aim of this dissertation is to verify the thesis that for a defined decision problem it is possible to define selection criteria and limitations that will allow the water system to maintain the greatest efficiency in the event of multiple failures. Based on the hydraulic model of the water supply network, information on the condition of equipment, consumables, data related to the number of available repair teams, information on the allocation of valves and the location of failures, the developed solution will allow to determine the order of their repair, achieving the intended goal function (minimizing the repair time in order to restoring the continuity of the water distribution process). The proposed solution is a solid foundation for Decision Support Systems and Expert Systems, because it is developed based on open software. This means that on the basis of the so-called Anyone can analyze, change, and distribute the modified version of a free software license.

The solution contains 7 algorithms, the combination of which will enable the implementation of the above-mentioned purpose. The task of the first proposed algorithm is to classify individual elements of the water supply network. This is to check the criticality and determine the importance factor of the element against the background of the entire collective water supply system. The second algorithm determines the so-called trace of water. Based on the location of the critical infrastructure, the path from the water source (water treatment plant or reservoir) to the node that represents the analyzed recipient is determined. The next algorithm, based on the data on the location of the shut-off valves, is responsible for determining the segments on the water supply network and the set of valves necessary for its insulation. The fourth algorithm is responsible for the classification of reported failures. Based on the specified factors, the failure is classified into one of the four available classes. The fifth algorithm, based on the results of the previous algorithms, gives particular failures (if necessary) a priority which is taken into account when scheduling repair orders. The task of the sixth algorithm is to identify

possible combinations of failures into one repair order. Based on the knowledge about the location of failures and their type, it is possible to aggregate failures in order to minimize the total number of valves necessary to close. The last algorithm is responsible for scheduling tasks. Based on the failure list, target function, and constraint function, a route is determined for each repair team and a list with a proposed sequence for repairing the failures.

The dissertation consists of nine chapters, a table of contents, a list of references, a list of figures, a list of tables, a list of acronyms and more important notations, and appendices. Each chapter begins with a short introduction to what is discussed in it the issue.

The first chapter (introduction) presents the motivation for the topic of the work. Then, the subject of the work (research problem), the purpose of the work and the content of the doctoral dissertation were defined.

The second chapter discusses selected modern control systems and monitoring. The general and layered control structure was presented and the methods of IT systems management in the operation process with the use of SCADA systems, Decision Support Systems and Expert Systems were described. In addition, this chapter deals with the integrity (interoperability) of complex IT systems.

The third chapter contains a theoretical introduction to the subject of collective water supply systems. This chapter provides basic information about the water treatment process, construction and types of water supply networks, and briefly describes the problem of criticality analysis of the elements included in such networks.

The fourth chapter is devoted to the methodology of modeling the water distribution process with the use of tools such as EPANET or WNTR. The chapter presents an example of a test modeling of a water network model, the topic of model calibration and two methods of modeling failures.

The fifth chapter is devoted to a review of the literature on IT solutions that may contribute to the improvement of the process of selecting the gate valves necessary to

separate the segment in which one or more failures occurred. The chapter also presents a selected method of selecting the valves to be closed with a slight modification.

The sixth chapter deals with the issues of making decisions about the order of repairing failures in water supply systems. The characteristics of failures that should be considered in the process of their classification were indicated and the prioritization process was discussed. In addition, this chapter is devoted to the characteristics of the problem of scheduling tasks

from the point of view of managing repair teams. The methods for solving task scheduling problems and the use of artificial intelligence algorithms, which can contribute to the improvement of the task scheduling process, are presented.

Chapter seven presents a group of algorithms that constitute the foundation of the decision support system, whose task is to improve the efficiency of the process of restoring water continuity in a post-catastrophic situation. The algorithms were developed on the basis of an analysis of the experience of a field expert from Miejskie Przedsiębiorstwo Energetyki Cieplnej Wodociągów i Kanalizacji in Środa Wielkopolska and a collection of scientific articles.

The eighth chapter contains the results of simulation studies on the proposed methodology for improving the efficiency of the process of restoring the continuity of water distribution in a post-catastrophic situation. In the first part of the chapter, the realized simulation environment was discussed and the prepared hydraulic models were presented.

In the next part, the generated test scenarios and the adopted assumptions are presented. The chapter ends with the results of the analysis of the operation and correctness of the proposed system.

The ninth chapter contains a summary and the final conclusions of the dissertation.

Wykaz akronimów i ważniejszych oznaczeń

α	-	*charakterystyka wycieku*
λ	-	*wskaźnik intensywności uszkodzeń*
Δt	-	*rozpatrywany okres (w latach)*
ρ	-	*gęstość cieczy*
A	-	*pole przekroju poprzecznego*
API	-	*Application Program Interface*
C_1	-	*klasa awarii mająca wysoki wpływ na infrastrukturę sieciową*
C_2	-	*klasa awarii mająca umiarkowany wpływ na infrastrukturę sieciową*
C_3	-	*klasa awarii mająca niewielki wpływ na infrastrukturę sieciową*
C_d	-	*współczynnik rozładowania*
CAQ	-	*Computer-Aided Quality*
CLP-FD	-	*Constaint Logic Programming over Finite Domain*
CMMS	-	*Computerised Maitenance Managgement System*
d_l	-	*zapotrzebowanie na wodę modelowanego wycieku*
D_1	-	*przewody o średnicy powyżej 300 mm*
D_2	-	*przewody o średnicy 100-300 mm*
D_3	-	*przewody o średnicy do 100 mm*
DD	-	*Demand Driven*
DIKW	-	*Łańcuch poznawczy: Data-to-Information-to-Knowledge-to-Wisdom*
Fx / faill_x	-	*awaria o identyfikatorze x*
F_{KP}	-	*współczynnik krytyczności przewodu*
F_S	-	*spadek przepływu (względem wyników wzorcowych)*
FCV	-	*Flow Control Valve*
FMECA	-	*Failure Modes and Effects Analysis*
g	-	*przyspieszenie ziemskie*
GPM	-	Gal per minute
GPV	-	*General Purpose Valve*
GUI	-	*Graphical User Interface*
h	-	*wysokość*
I_T	-	*Todini index*
IKx	-	*budynek infrastruktury krytycznej o indeksie x*
IUR	-	*Inżynieria Utrzymania Ruchu*
J_{BW}	-	*liczba węzłów w sieci pozbawiona dostępu do wody*
J_{CL}	-	*liczba wszystkich węzłów w sieci*
J_{OC}	-	*liczba węzłów w sieci, z obniżonym ciśnieniem*
L	-	*długość (w km) analizowanych przewodów w okresie* (Δt)
Lx	-	*poziom zleceń o identyfikatorze x*

LPS	-	Litr per secound
MES	-	*Manufacturing Execution System*
MPC	-	*Model Predictive Control*
$n\ (\Delta t)$	-	*liczba awarii w okresie* (Δt)
O	-	*ochrona systemu przed wystąpieniem awarii*
OSM	-	*Open Street Map*
p	-	*średni nacisk, nadciśnienie*
P_{DPW}	-	*procentowy udział w transporcie wody*
P	-	*prawdopodobieństwo wystąpienia awarii*
PAC	-	*Programmable Automation Controller*
PBV	-	*Pressure Breaker Valve*
PDD	-	*Pressure Demand Driven*
PE	-	*Polietylen*
PLC	-	*Programmable Logic Controller*
PSV	-	*Pressure Stabilizing Valve*
PRV	-	*Pressure Reducing Valve*
PVC	-	*Polichlorek winylu*
PZH	-	*Państwowy Zakład Higieny*
r_{safe}	-	*szacownie ryzyka z punktu widzenia „safe”*
$r_{security}$	-	*szacownie ryzyka z punktu widzenia „security”*
RTx	-	*ekipa naprawcza o identyfikatorze x*
S	-	*straty wynikające z tytułu wystąpienia awarii*
Sx	-	*segment o identyfikatorze x*
SCADA	-	*Supervisory Control And Data Acqusition*
SDW	-	*System Dystrybucji Wody*
SE / ES	-	*Systemy Ekspertowe / Expert Systems*
SUW	-	*Stacja Uzdatniania Wody*
SWD / DSS	-	*System Wspomagania Decyzji / Decision Support System*
SZZwW	-	*System Zbiorowego Zaopatrzenia w Wodę*
T_{OnN}	-	*czas oczekiwania na naprawę*
T_{PN}	-	*przewidywany czas naprawy awarii*
T_{PP}	-	*czas potrzebny na przygotowanie podłoża*
T_{RN}	-	*rzeczywisty czas naprawy awarii*
T_{ZP}	-	*czas potrzebny na zabezpieczenie podłoża*
TCV	-	*Throttle Control Valve*
V	-	*podatność systemu na wystąpienie awarii*
Wx	-	*magazyn o identyfikatorze x*
WNTR	-	*Water Network Tool for Resilience*
WOD – KAN	-	*Przedsiębiorstwo Wodno-Kanalizacyjne*

Spis treści

Streszczenie **5**

Abstract **9**

Wykaz akronimów i ważniejszych oznaczeń **13**

1. Wstęp **17**

1.1 Wprowadzenie 17

1.2 Przedmiot, cele i teza rozprawy 19

2. Wybrane systemy sterowania i monitorowania **23**

2.1 Wprowadzenie 23

2.2 Warstwowa struktura systemów sterowania 24

2.3 Systemy Wspomagania Decyzji oraz Systemy Ekspertowe 27

2.4 System nadzorowania, monitorowania i sterowania SCADA 32

2.5 Integracja złożonych systemów informatycznych 34

2.6 Przykłady zintegrowanych systemów informatycznych w inżynierii środowiska 35

3. Charakterystyka sieci wodociągowych **39**

3.1 Wprowadzenie 39

3.2 Sieci wodociągowe – przebieg procesu dystrybucji wody 40

3.2.1 Budowa sieci wodociągowych 40

3.2.2 Rodzaje sieci wodociągowych 45

3.2.3 Sposoby wykrywania i naprawy awarii 49

3.2.4 Analiza krytyczności sieci wodociągowej 50

4. Modelowanie procesu dystrybucji wody **55**

4.1 Wprowadzenie 55

4.2 Narzędzia informatyczne wspomagające modelowanie procesu dystrybucji wody 57

4.2.1 EPANET 57

4.2.2 The Water Network Tool for Resilience (WNTR) 59

4.3 Modelowanie sieci wodociągowych 60

4.3.1 Charakterystyka przykładowego modelu sieci wodociągowej 61

4.3.2 Modelowanie sieci wodociągowej z wykorzystaniem środowiska EPANET 68

4.3.3 Modelowanie sieci wodociągowej z wykorzystaniem biblioteki WNTR 76

4.4 Kalibracja modeli hydraulicznych systemów dystrybucji wody 80

4.5 Modelowanie awarii 81

5. Metodyka typowania zasuw do zamknięcia stosowana w modelowaniu matematycznym systemów wodociągowych **87**

5.1 Wprowadzenie 87

5.2 Sposoby modelowania zasuw odcinających 88

5.3 Algorytm typowania zasuw do zamknięcia 91

5.3.1 Podejście macierzowo-grafowe 92

5.3.2 Podejście macierzowo-grafowe z analizą przepływów 99

6. Podejmowanie decyzji o kolejności usuwania awarii w systemach wodociągowych **103**

6.1 Wprowadzenie 103

6.2 Klasyfikacja awarii 104

6.3 Priorytetyzacja 108

6.4 Algorytmy szeregowania zadań w inżynierii środowiska 109

6.4.1 Metodyka rozwiązywania problemów szeregowania 110

6.4.2 Wykorzystanie algorytmów szeregowania zadań w inżynierii środowiska 112

7. Efektywne algorytmy przywracania ciągłości dostawy wody ... 115
7.1 Wprowadzenie ... 115
7.2 Algorytm klasyfikacji elementów sieci wodociągowej ... 115
7.3 Algorytm wyznaczania tras przepływu wody (cel – źródło) ... 122
7.4 Algorytm typowania zasuw do zamknięcia z analizą przepływów ... 126
7.5 Algorytm klasyfikacji awarii ... 128
7.6 Algorytm priorytetyzacji ... 135
7.7 Algorytm agregacji awarii ... 140
7.8 Algorytm szeregowania zadań dostępnych ekip remontowych ... 143
8. System Wspomagania Decyzji w procesie przywracania ciągłości dystrybucji wody po wystąpieniu sytuacji katastroficznej ... 157
8.1 Wprowadzenie ... 157
8.2 Środowisko symulacyjne ... 157
8.3 Charakterystyka wybranych modeli hydraulicznych ... 158
8.3.1 Model hydrauliczny 1 – Net3 ... 159
8.3.2 Model hydrauliczny 3 – Środa Wielkopolska ... 161
8.4 Przykładowe scenariusze zdarzeń katastroficznych (dane wejściowe) ... 163
8.4.1 Scenariusze testowe – model Net3 ... 164
8.4.2 Scenariusze testowe – model Środy Wielkopolskiej ... 171
8.5 Poprawność działania systemu i analiza wyników ... 180
9. Wnioski ... 185
9.1 Podsumowanie ... 185
9.2 Wnioski końcowe ... 187
Bibliografia ... 191
Spis rysunków ... 199
Spis tabel ... 201
Załączniki ... 203
Załącznik A – Kod programu analizującego przepływy wraz z wynikami przykładowego scenariusza testowego ... 203
Załącznik B – Plik konfiguracyjny (config.py) ... 207
Załącznik C – Kod źródłowy Algorytmu 1 ... 210
Załącznik D – Kod źródłowy Algorytmu 2 ... 216
Załącznik E – Informacje o zasuwach w przykładowym modelu testowym ... 218
Załącznik F – Kod źródłowy Algorytmu 3 ... 219
Załącznik G – Informacje o awariach w przykładowym modelu testowym ... 223
Załącznik H – Informacje dodatkowe o analizowanej sieci ... 228
Załącznik I – Kod źródłowy Algorytmu 4 ... 229
Załącznik J – Kod źródłowy Algorytmu 5 ... 233
Załącznik K – Kod źródłowy Algorytmu 6 ... 235
Załącznik L – Zawartość scenariusza testowego ... 237
Załącznik M – Kod źródłowy Algorytmu 7 ... 238

1. Wstęp

1.1 Wprowadzenie

Woda jest niezbędnym zasobem w każdym aspekcie ludzkiego życia. Od jej jakości oraz dostępności uzależniona jest szansa na zdrowe i godne życie. Trudności w dostępie do wód mają bezpośredni wpływ na możliwość edukacji i rozwoju człowieka, a także rozwój gospodarczy miast czy państw. Pomimo faktu, że zajmuje ona około 70.8% powierzchni całej kuli ziemskiej (w postaci standardowej, tzn. ciekłej), tylko niewielka jej część jest zdatna do spożycia. Woda słodka (około 3.5 %) to przede wszystkim śniegi i lodowce, których lokalizacja uniemożliwia ich szersze rozdysponowanie i wykorzystanie. Najczęstszymi źródłami wody słodkiej wykorzystywanymi przez człowieka są więc rzeki, jeziora oraz wody podziemne. Jednakże, ze względu na wzrost liczby ludności oraz nieustanny rozwój przemysłu i urbanistyki stan tych wód systematycznie ulega pomniejszeniu. W tym celu konieczne jest prowadzenie badań z zakresu uzdatniania, które przyczyniają się do wzrostu wiedzy i świadomości ludzi w tematyce przebiegu procesów prowadzących do otrzymania wody zdatnej do spożycia oraz jej magazynowania. Sytuacja ta i świadomość, jak woda jest cennym zasobem przyczyniła się do powstania ponadczasowej myśli (de Saint-Exupéry): *„Woda – nie ma ani smaku, ani koloru, ani zapachu. Nie można jej opisać. Pije się ją, nie znając jej. Nie jest niezbędna do życia – jest samym życiem”* [71, 79].

Wiedząc jak ważna jest woda, naturalnym zadaniem wydaje się utrzymanie ciągłości jej dystrybucji z zachowaniem wszelkich norm jakości. Zbiór norm i reguł dotyczących jakości wody oraz sposobów zapewnienia odpowiedniej ilości wody pod właściwym ciśnieniem określony został w Rozporządzeniu Ministra Zdrowia z dnia 29 marca 2007 roku oraz w Ustawie z dnia 7 czerwca 2011 r. o zbiorowym zaopatrzeniu w wodę i zbiorowym odprowadzaniu ścieków [78, 96]. Ponadto, zgodnie z założeniami zrównoważonego rozwoju opublikowanymi przez Światową Komisję ds. Środowiska i Rozwoju ONZ w 1987 roku strategia eksploatacji zasobów i infrastruktury wodnej powinna funkcjonować w taki sposób, aby zaspokoić potrzeby obecne, nie zagrażając możliwości zaspokojenia potrzeb przyszłych pokoleń [15, 61]. W celu zapewnienia najwyżej jakości wody w procesie uzdatniania wykorzystuje się Stacje Uzdatniania Wody (SUW).Zadaniem tego typu stacji jest między innymi [5]: ujmowanie wody, usuwanie

zanieczyszczeń mechanicznych, usuwanie związków żelaza i manganu, kontrola twardości wody, dezynfekcja oraz wiele innych. Poza dbaniem o jakość wody zadaniem Stacji Uzdatniania Wody jest również współpraca z Przedsiębiorstwem Wodno-Kanalizacyjnym (WOD-KAN) w celu nadzorowania i zarządzania całą infrastrukturą wodociągową. Przez infrastrukturę wodociągową rozumie się, zespół obiektów technicznych oraz urządzeń, których zadaniem jest ujmowanie, uzdatnianie, monitorowanie stanu, magazynowanie, transport oraz dystrybucja wody [19, 33]. Za transport oraz dystrybucję wody w całej infrastrukturze sieciowej odpowiada sieć wodociągowa, która jest następstwem akweduktów. Akwedukt (łac. aquae ductus) oznacza dosłownie ciąg wodny tj. kanał wodny (podziemny lub naziemny), który wykorzystując siły grawitacji doprowadza wodę do miasta z oddalonego od niego źródła [19]. Pomimo faktu, że pierwszy akwedukt rzymski powstał w 312 roku p.n.e., jego cel (tj. dostarczenie wody ze źródła do odbiorcy) dzisiejszych sieci wodociągowych pozostał niezmienny. Zauważyć można jednak wiele różnic w budowie i funkcjonowaniu obecnych sieci. Rozwój techniczno-technologiczny umożliwił eksploatacje elementów wykonawczych (np. zespołu pomp) do transportu wody oraz wykorzystanie nowoczesnych systemów monitorowania i nadzorowania sieci, które w sposób ciągły monitorują stan sieci jako infrastruktury budowlanej oraz medium, które się w niej znajduje.

Ze względu na swoją budowę (rozpiętość na dużym obszarze) oraz niezbędne do poprawnego funkcjonowania jednostek i mas zadanie, jakim jest dystrybucja wody – sieć wodociągowa uważana jest za jeden z najbardziej newralgicznych elementów całej infrastruktury wodociągowej [19, 46, 71]. Nie chodzi tutaj tylko o kwestie bezpieczeństwa (woda jest przecież jednym ze strategicznych surowców, co oznacza możliwość ataku chemicznego lub biologicznego, a co za tym idzie chorobę i śmierć mieszkańców skażonego obszaru), ale również kwestia stałego dostępu do wody. Jak wskazują badania prowadzone z zakresu psychologii kognitywnej i zachowania ludzi w sytuacjach kryzysowych, w przypadku dłuższej przerwy w dostępie do podstawowych mediów (takich jak woda czy prąd) istnieje wysokie prawdopodobieństwo wystąpienia zakłócenia stanu równowagi jednostki z otoczeniem [46, 68]. Gdy zachodzi pewnego rodzaju dysonans między obecną potrzebą, a możliwością jej zaspokojenia organizm człowieka zaczyna odczuwać pewnego rodzaju zagrożenie, co może skutkować jego

niepożądanym, a często również niebezpiecznym zachowaniem. W sytuacji, gdy problem ten przestaje dotyczyć tylko pojedynczych osób, a zaczyna odnosić się do większej liczby ludzi – sytuacja staje się bardziej poważna. Ważne jest więc, aby zachować ciągłość procesu dystrybucji wody. Niezmienność tej sytuacji spowoduje zaspokojenie podstawowej potrzeby fizjologicznej oraz nie narazi na powstawanie sytuacji stresogennych.

W przypadku wystąpienia awarii na sieci wodociągowej zadaniem ww. podmiotów (tj. stacji uzdatniania wody oraz przedsiębiorstwa wodno-kanalizacyjnego) jest jak najszybsze określenie miejsca jej występowania, wyznaczenie zespołu odpowiedzialnego za jej naprawę, zabezpieczenie miejsca awarii oraz naprawa skutkująca przywróceniem procesu dystrybucji wody na wskazanym obszarze. O ile przywrócenie pełnej sprawności sieci wodociągowej przy małej liczbie awarii i ograniczonych zasobach nie wydaje się trudnym zadaniem, o tyle perspektywa wystąpienia sytuacji kryzysowej (np. wielu awarii jednocześnie) może stanowić problem. W naukach społecznych wyróżniono wiele czynników powodujących powstanie sytuacji kryzysowych oraz teorii ich przebiegu. Według nich spowodowane są one niewłaściwymi działaniami rutynowymi lub są następstwem awarii technicznych lub katastrof naturalnych [12, 88, 89, 102]. W przypadku wyżej wymienionych sytuacji (np. wstąpienie klęski żywiołowej jak trzęsienie ziemi o dużej amplitudzie drgań wstrząsów sejsmicznych), sieć wodociągowa jest szczególnie narażona na uszkodzenia. W sytuacji wystąpienia takich okoliczności powoływane są specjalne zespoły zarządzania kryzysowego (składające się z osób oraz podmiotów), które w sposób skoordynowany będą zaangażowane w akcję przywrócenia sprawności systemu wodociągowego.

1.2 Przedmiot, cele i teza rozprawy

Pomimo szybkiego rozwoju techniczno-technologicznego, wiedzy i świadomości o procesach zachodzących w środowisku trudno jest przewidzieć: kiedy, gdzie i w jakim stopniu może wystąpić katastrofa lub klęska żywiołowa. Niezależnie od znajomości przyczyny powstawania kataklizmów nikt nie jest w stanie ich wyeliminować – jesteśmy jedynie w stanie ograniczyć ich skutki. Modelowanie pracy obiektów wodociągowych w sytuacjach wymienionych powyżej wraz z ograniczeniami typu dostępność zasobów,

umożliwia testowanie złożonych algorytmów szeregowania i sterowania pracą dostępnych ekip naprawczych.

Przedmiotem badań opisanych w niniejszej rozprawie doktorskiej są algorytmy klasyfikacji awarii oraz szeregowania zadań ekip naprawczych. Większość systemów zaopatrzenia w wodę wyposażona jest w systemy monitorowania, archiwizacji danych oraz sterowania przebiegiem procesu. Na podstawie danych otrzymanych z ww. systemów, nadzorca (ekspert, operator) posiada wiedzę na temat stanu procesu co umożliwia mu wpływ na jego dalszy przebieg. Jednak w przypadku wystąpienia sytuacji katastroficznej, a co za tym idzie wielu awarii jednocześnie, wiedza eksperta może nie wystarczyć, aby w dostatecznie krótkim czasie przywrócić ciągłość procesu dystrybucji wody. Ważne jest więc to, aby współtworzyć systemy wspomagania decyzji w oparciu o najnowsze rozwiązania techniczno-technologiczne, których zadaniem będzie wsparcie operatora w podjęciu decyzji. To ważne, aby współczesne systemy dystrybucji wody (SDW), ze względu na swoją złożoność i reprezentatywność korzystały z zaawansowanych systemów wykorzystujących złożone algorytmy sterowania [15, 91].

Tematyka rozprawy analizuje współczesne problemy szeregowania zadań na przykładzie modelowania sytuacji kryzysowych występujących na sieci wodociągowej. Praca dotyczy trzech dyscyplin naukowych: automatyka, elektronika i elektrotechnika, informatyka techniczna i telekomunikacja oraz inżynieria środowiska, górnictwo i energetyka. Omawiana problematyka skupia się na dynamicznym wyznaczaniu zadań dostępnym ekipom naprawczym w oparciu o informację uzyskane z modelu matematycznego sieci wodociągowej oraz przyjętych kryteriach.

Celem pracy było opracowanie grupy algorytmów stanowiących fundament systemu wspomagania decyzji, których zadaniem jest poprawa efektywności procesu przywracania ciągłości wody w sytuacji po katastroficznej. W oparciu o różne modele hydrauliczne sieci wodociągowej (różne pod względem: struktury, liczby węzłów i przewodów), analizie zostały poddane opracowane scenariusze katastroficzne. Poprzez scenariusz katastroficzny rozumie się, wystąpienie sytuacji wielu awarii na wybranym modelu sieci wodociągowej. Scenariusz określa informacje takie jak: liczba awarii i ich typ, liczba dostępnych ekip naprawczych i ich lokalizacja początkowa, stan magazynowy

dostępnych urządzeń i części zamiennych, lokalizacja magazynu/ów, informacja o lokalizacji budynków krytycznych (np. szpitale), itp. Punktem wyjściowym do opracowania metodyki była analiza doświadczeń eksperta dziedzinowego z *Miejskiego Przedsiębiorstwa Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej* oraz zbiór artykułów naukowych. W oparciu o przetworzone informacje sformułowano ograniczenia oraz kryteria sterowania.

W ramach opracowanej metodyki zaproponowano 7 algorytmów stanowiących podstawę systemu wspomagania decyzji i/lub systemu wczesnego ostrzegania. Zadaniem algorytmów była analiza sytuacji kryzysowej i wsparcie zespołu zarządzania kryzysowego w podjęciu decyzji o kolejności naprawy awarii, w tym zarządzanie dostępnymi zasobami. Powyższy zakres pracy stanowi zadania szczegółowe (wspierające realizację celu głównego pracy), który ma udowodnić sformułowaną tezę pracy:

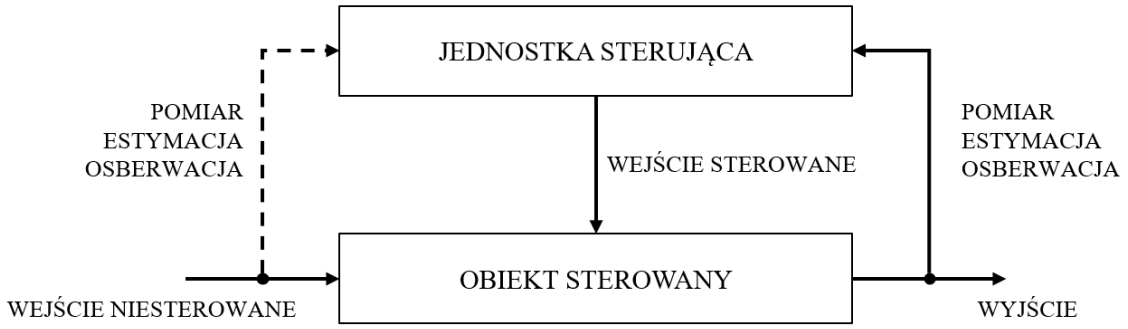
> ***„Dla zdefiniowanego problemu decyzyjnego możliwe jest określenie kryteriów wyboru, ograniczeń, które pozwolą zachować największą sprawność systemu wodociągowego w sytuacji wystąpienia wielu awarii”.***

Celem udowodnienia tezy, zaproponowana metodyka została zrealizowana w postaci modułowego systemu informatycznego, integrującego modele matematyczne sieci wodociągowych, scenariusze testowe, informacje dodatkowe o sieci wodociągowej (np. o lokalizacji zasuw) oraz zaimplementowane algorytmy. Tego typu środowisko umożliwiło przeprowadzenie badań opracowanych algorytmów dla kilku wybranych modeli hydraulicznych sieci wodociągowej.

2. Wybrane systemy sterowania i monitorowania

2.1 Wprowadzenie

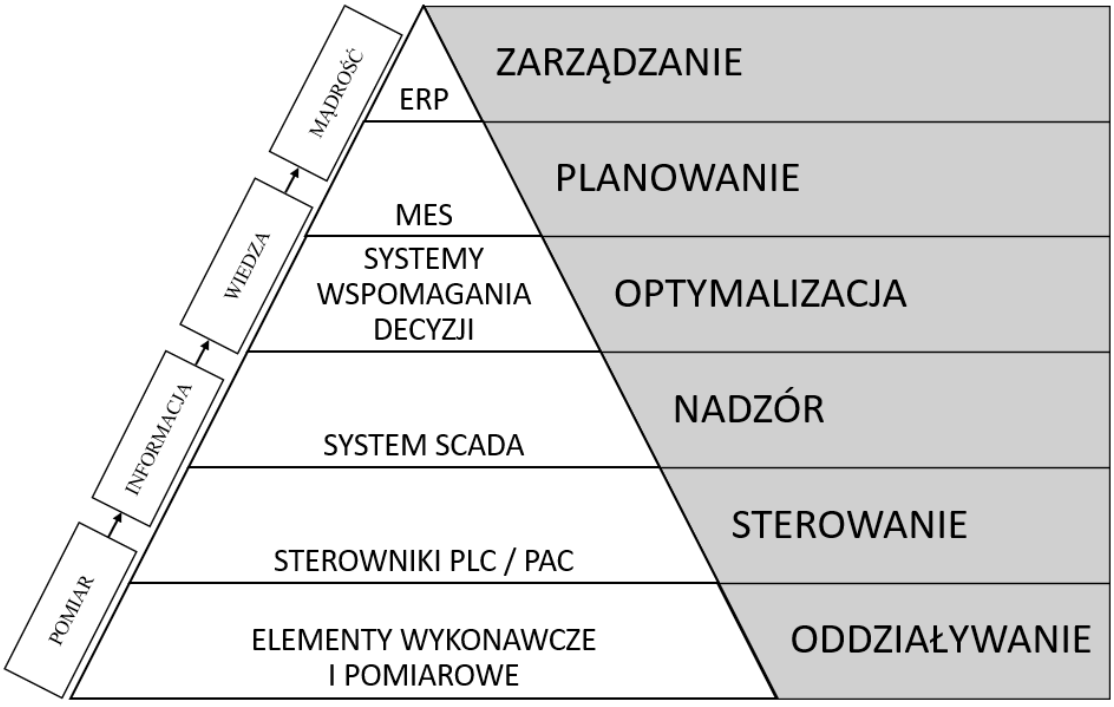
W szerokim ujęciu mianem sterowania określa się wykonywanie zamierzonych czynności, mających wpływ na przebieg procesu. Dokonać tego można w sposób automatyczny bądź ręczny. W przypadku sterowania automatycznego sygnały sterujące (inaczej: wielkości sterujące) przetwarzane są za pośrednictwem urządzeń wykonawczych i ingerują w (wpływają na) proces. Wyróżnia się dwie kategorie sterowania: w układzie otwartym oraz zamkniętym. W przypadku układu zamkniętego, cechą charakterystyczną jest obecność sprzężenia zwrotnego (brak tej cechy w przypadku układu otwartego), które umożliwia porównanie wartości zadanej i wartości zmierzonej wielkości sterowanej [93]. Proces, który podlega sterowaniu zachodzi w tzw. obiekcie sterowanym. Obiekt sterowany jest wyizolowanym fragmentem środowiska, w którym się znajduje. Jego stan uzależniony jest od wpływu (kontrolowanego i niekontrolowanego) otoczenia. W oparciu o ocenę wartości wyjścia obiektu sterowanego, wejścia sygnałów niekontrolowanych (tj. zakłóceń, które określane są na podstawie pomiarów bądź estymacji) oraz wartości zadanej określa się poprawność oddziaływania wejścia sterowania na obiekt. Za wytworzenie sygnału sterującego, odpowiada jednostka sterująca (np. mikrokontroler lub sterownik), którego zadaniem jest utrzymanie lub zmiana wartości wejść sterowanych [2, 90]. Rys. 2.1 przedstawia ogólną strukturę systemu sterowania.



Rys. 2.1 Ogólna struktura systemów sterowania [2, 90]

2.2 Warstwowa struktura systemów sterowania

Współczesne systemy sterowania poza utrzymaniem wielkości sterowanej w otoczeniu wartości zadanej powinny realizować również inne zadania typu [93]: archiwizacja i interpretacja danych na temat stanu analizowanego procesu, łatwy i szybki dostęp do dużej ilości danych, możliwość stosowania złożonych algorytmów sterowania oraz uwzględnienie korelacji między zmiennymi procesowymi. Większość procesów (obiektów sterowanych) ze względu na swoją złożoność nie jest w stanie zrealizować celu głównego. Aby zrealizować złożony cel główny konieczna jest jego dekompozycja na cele cząstkowe. Dopiero ich realizacja pozwoli osiągnąć cel podstawowy [90]. W celu realizacji zadań cząstkowych aktualne systemy sterowania projektuje się w postaci hierarchicznej (warstwowej). Warstwowa struktura sterowania powstaje w wyniku dekompozycji funkcjonalnej, w której obiekt sterowany rozpatrywany jest jako całość, z którego wydzielić można powiązane ze sobą zadania (cele) cząstkowe. W poszczególnych warstwach podejmowane są decyzje, które całościowo wpływają na ten sam obiekt [2, 90].



Rys. 2.2 Warstwowa struktura systemów sterowania [2, 3, 90, 113]

Rys. 2.2 przedstawia przykładową warstwową strukturę systemu sterowania. Celem tego typu podejścia jest przede wszystkim zachowanie jakości wytwarzanego produktu (woda zdatna do spożycia), poprawa efektywności działania obiektu sterowania oraz minimalizacja zakłóceń i awarii zapewniając poprawny przebieg procesu dystrybucji wody.

Warstwa oddziaływania składa się z elementów pomiarowych i wykonawczych. Zadaniem elementów pomiarowych jest uzyskanie informacji na temat stanu procesu. Odbywa się to dzięki przetwarzaniu sygnałów odebranych z czujników (najczęściej realizowane jest to w czasie rzeczywistym). Urządzenia wykonawcze natomiast mają za zdanie wypracowanie sygnału wejściowego na obiekt sterowania na podstawie sygnałów sterujących [2, 93]. W przypadku systemów wodociągowych do tej warstwy możemy zaliczyć urządzenia takie jak: pompa, zasuwa, wodomierz, ciśnieniomierz, itp.

Warstwa sterowania bezpośredniego (sterowniki programowalne PLC lub PAC) nadzoruje aktualny stan procesu wykorzystując metodę sterowania bezpośredniego lub operacyjnego. W sytuacji sterowania bezpośredniego wartość zmierzona zostaje przekształcona tak, aby była reprezentacją liczbową wielkości sterowanej. Następnie na podstawie wartości zadanej i wartości zmierzonej wyznaczony zostaje uchyb regulacji. W oparciu o nową wartość uchybu wyznaczana jest nowa wartość sygnału sterującego. Przykładami wykorzystywanych algorytmów sterowania w tej warstwie mogą być algorytmy klasyczne (PI, PD, PID), adaptacyjne, predykcyjne (ang. *Model Predictive Control*) lub wykorzystujące elementy sztucznej inteligencji (sztuczne sieci neuronowe, logika rozmyta czy algorytmy ewolucyjne).

Warstwa nadzoru (warstwa sterowania nadrzędnego) monitoruje wielkości procesowe, odpowiadające za jakość dystrybuowanej wody. Systemy zawarte w tej warstwie mogą być stale rozwijane na podstawie wiedzy i doświadczenia zdobytego podczas sterowania obiektem. Ponadto systemy sterowania nadrzędnego w oparciu o złożone algorytmy mogą przewidywać stan nadzorowanego procesu (w oparciu o dane historyczne) oraz dokonywać estymacji aktualnych wartości zmiennych procesowych, których wartości nie są wyznaczane w sposób bezpośredni [3, 93]. Przykładem takiego systemu jest system SCADA (szerzej opisany w Rozdziale 2.4).

Większość z obecnie wdrażanych systemów sterowania procesami gospodarki wodno-ściekowej, kończy się na warstwie nadzoru i systemie SCADA. Oznacza to, że warstwy te nie są w żaden sposób połączone z warstwą planowania i zarządzania całego przedsiębiorstwa. Skutkiem braku połączenia tych warstw jest utrata wielu cennych informacji, które mogłyby przyczynić się do wzrostu wiedzy operatora oraz w razie potrzeby pomóc mu w podjęciu decyzji. Warto więc w trakcie planowania zaproponować warstwę optymalizacji, która mogła by [113]:

- minimalizować koszty całkowite przedsiębiorstwa z zachowaniem jakości wody określonej w odpowiednich rozporządzeniach (cel ekonomiczny);
- maksymalizować jakość wody, przy zachowaniu dopuszczalnych kosztów (cel ekologiczny);
- wpłynąć na efektywne zarządzanie – oszczędność czasu oraz materiałów, minimalizację kosztów usuwania awarii, bezpieczeństwo realizowanych prac czy świadome planowanie przyszłych przedsięwzięć (cel organizacyjny).

Jako ograniczenia do powyższych funkcji celu można przyjąć: wymaganą jakość wody, aktualną wydajność elementów wykonawczych, dopuszczalne zużycie materiałów, półproduktów czy energii [109, 113].

Warstwa planowania w warstwowej strukturze systemów sterowania odpowiada za realizację planów operacyjnych. Na podstawie danych otrzymanych w wyniku monitorowania procesu określa się: terminy przeglądów technicznych urządzeń pomiarowych i wykonawczych, kalibrację urządzeń pomiarowych, zakup części zamiennych i materiałów eksploatacyjnych, kolejność usuwania awarii oraz przydział ekip naprawczych. Do systemów informatycznych wspierających warstwę planowania możemy zaliczyć [3]: MES (ang. *Manufacturing Execution Systems*), CMSS (ang. *Computerised Maintenance Management System*) czy CAQ (ang. *Computer Aided Quality*).

Najwyższą warstwą w hierarchicznym systemie sterowania jest warstwa zarządzania, w skład której wchodzą systemy wspomagające proces zarządzania

zasobami przedsiębiorstwa. Najważniejszą rolą tej warstwy jest szeroko rozumiana analiza, wspomaganie decyzji biznesowych, planowanie operacyjne, kontrola realizacji przyjętej strategii działania, przygotowanie planu modernizacji, szacowanie możliwości zwiększenia produkcji, analiza rentowności czy ograniczenie kosztów [3, 113].

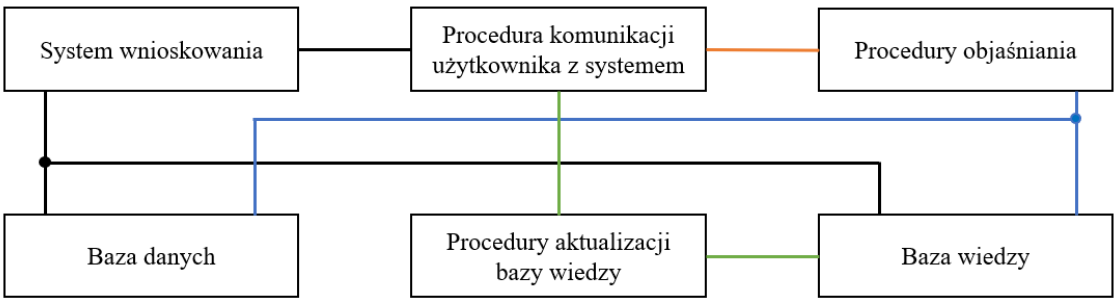
Na Rys. 2.2 przedstawiony został również łańcuch poznawczy DIKW (ang. *Data to Information to Knowledge to Wisdom*), którego zadaniem jest uświadomienie jak cenny jest każdy pomiar. Wraz z kolejną warstwą pomiar ewoluuje w informację, która przeradza się w wiedzę i mądrość nadzorcy procesu. Zależności między poszczególnymi ogniwami wykorzystywane są w procesach zarządzania wiedzą, która pozwala na podejmowanie decyzji oraz poprawę działania całego procesu [3].

2.3 Systemy Wspomagania Decyzji oraz Systemy Ekspertowe

Głównym zadaniem współczesnych systemów sterowania nie jest już wyłącznie analiza danych oraz nadzór nad przebiegiem procesu. Coraz częściej w wymaganiach systemowych pojawia się sformułowanie informujące o funkcji doradczej systemu. Oznacza to wsparcie pracy operatora w dziedzinie sterowania przebiegiem procesu, poprzez określenie odpowiednich wartości zadanych dla warstwy sterowania bezpośredniego i nadrzędnego [90]. Najczęściej w celu optymalizacji punktu pracy wykorzystuje się złożone algorytmy sterowania predykcyjnego MPC (ang. *Model Predictive Control*) wykorzystujące statyczne modele liniowe (wadą tego typu systemów jest brak możliwości rozpatrywania procesów nieliniowych) [3, 53, 108]. W rzeczywistości, często okazuje się, że nadzorca procesu decyduje się na zmianę wartości zadanej. Spowodowane jest to jego wiedzą, którą zdobywał na przestrzeni lat, lub instrukcją postępowania określającą zespół czynności potrzebnych do wykonania pewnego zadania. Decyzja podjęta przez operatora powoduje uzyskanie zamierzonego celu, jednak nie zawsze odbywa się to w sposób efektywny. Przykładem systemów proponujących rozwiązanie na podstawie analizy danych i wiedzy operatora są Systemy Wspomagania Decyzji (ang. *Decision Suport System*). Zadaniem tego typu rozwiązania jest wsparcie lub zastąpienie złożonych procesów rozumowania [3]. Jako inny przykład warto wymienić również Systemy Ekspertowe (ang. *Expert System*), które zgodnie z definicją [43] oznaczają *program komputerowy, realizujący zadania wymagające inteligencji. Wynik wykonania zadania powinien być zbliżony do wyniku osiągniętego*

przez człowieka realizującego to zadanie. Systemy ekspertowe możemy podzielić na systemy: doradcze (ang. *advisory*), niezależne (ang. *dicattorial*) oraz krytykujące (ang. *criticizing*). W przypadku systemów doradczych wynikiem działania jest przykład rozwiązania, które operator procesu może zaakceptować bądź odrzucić. Systemy niezależne, podejmują decyzję bez ingerencji nadzorcy procesu, natomiast krytykujące informują operatora o powstałym problemie wraz z przykładowym rozwiązaniem podając sposób rozumowania (poprzez analizę i komentarz) [57].

Za projektowanie i realizację systemów ekspertowych odpowiadają tzw. Inżynierowie Wiedzy. Ich zadaniem jest uzyskanie i odpowiednie ustrukturyzowanie wiedzy przekazanej przez nadzorcę lub nadzorców procesu, w sposób zrozumiały dla maszyn [56]. Tak zarchiwizowana i ustrukturyzowana wiedza eksperta dziedzinowego nosi nazwę bazy wiedzy (ang. *knowledge database*). Baza wiedzy umożliwia pracę systemu wspomagania decyzji lub systemu ekspertowego bez konieczności obecności eksperta. Ponadto, udowodniono, że poprawność i efektywność tego typu systemów zależy przede wszystkim od wiedzy w nim zawartej [57, 62]. Reprezentując systemy eksperckie jako bardziej złożoną strukturę elementów wzajemnie na siebie oddziaływujących możemy wyróżnić [3, 57]: bazę danych (ang. *data bases*), bazę wiedzy (ang. *knowledge database*), system wnioskujący (ang. *inference system*), procedury opisujące sposób komunikacji użytkownika z systemem oraz procedury rozwoju systemu. Na Rys. 2.3 przedstawiono przykładową strukturę systemu ekspertowego wraz z oddziaływaniem poszczególnych elementów na resztę systemu.



Rys. 2.3 Struktura systemu eksperckiego [3, 57]

Budowa systemu wspomagania decyzji lub systemu eksperckiego realizowana może być w sposób dedykowany (od podstaw, w celu realizacji konkretnego systemu) bądź szkieletowy, w którym na podstawie gotowej implementacji systemu uzupełnia się bazę wiedzy. Ze względu na realizowane zadania możemy wyróżnić systemy [57]: predykcyjne, interpretacyjne, diagnostyczne, monitorowania, planowania, sterowania oraz instruowania.

Jak już wspomniano, baza wiedzy jest najważniejszą częścią systemu ekspertowego. Wiedza otrzymana od nadzorcy analizowanego procesu jest zbiorem informacji dotyczącej pewnej dziedziny, powstałym w wyniku gromadzenia się doświadczeń oraz przebiegu procesu uczenia się [3, 62]. W przypadku logiki wiedza definiowana jest natomiast jako zbiór reguł oraz faktów. Istnieje wiele metod reprezentacji wiedzy. Najczęściej metody te dzielimy na symboliczne i niesymboliczne. W skład reprezentacji symbolicznej wchodzi reprezentacja deklaratywna (charakteryzująca się określaniem tzw. zbiorów specyficznych dla znanych faktów lub reguł) oraz reprezentacja proceduralna (cechująca się określeniem zbioru procedur). W przypadku metod symbolicznej reprezentacji wiedzy wykorzystuje się najczęściej [3, 62]:

- stwierdzenia – odnoszące się bezpośrednio do objawów, czynności oraz zdarzeń, które zapisywane są w strukturze tzw. trójki lub czwórki uporządkowanej (np. [obiekt, atrybut, wartość] lub [obiekt, atrybut, wartość, stopień pewności], gdzie wartość stopnia pewności ustalana jest w sposób subiektywny z określonego zakresu);
- rachunek zdań i predykatów – w których minusem jest tworzenie nadmiarowych tautologii (ponieważ, w przypadku gdy mamy zdanie A∧B, które jest prawdziwe, to prawdą jest również A∧B∧B oraz A∧B∧B∧B);
- reguły – wykorzystujące strukturę IF *przesłanka* THEN *działanie*. Zaletą reguł jest brak występowania systemu wnioskującego, ponieważ wszystkie reguły zawierają w działaniu wynik końcowy;
- wektory wiedzy – są prostszą wersją reguł opisanych powyżej, dla których stosuje się zapis wektorowy. W przypadku wektorów wiedzy wykorzystuje się symbole reprezentacji wiedzy takie jak: T oznaczające prawdziwość

warunku, F oznaczające fałszywość warunku oraz * oznaczająca brak warunku w regule.

Warto dodać, że symboliczna reprezentacja wiedzy wykorzystuje modele obliczeniowe, natomiast reprezentacja niesymboliczna charakteryzuje się opisem doświadczeń (w tym obserwacji) zebranych z otoczenia. Najczęściej wykorzystywanymi metodami reprezentacji w tym zakresie są algorytmy ewolucyjne, sztuczne sieci neuronowe oraz systemy uczące się [3].

W przypadku tworzenia baz wiedzy, inżynierowie wiedzy najczęściej wykorzystują narzędzia do edycji tekstu, które w łatwy i klarowny sposób umożliwiają wprowadzanie danych. Ze względu na strukturę przechowywanych informacji możemy wyróżnić: bazę tekstową (np. słownik), bazę danych, bazę reguł (przechowującą informację o zależnościach) oraz bazę wiedzy rozsądkowej (zawierająca informację o zachowaniu człowieka w przypadku wystąpienia zdarzenia) [3, 62].

Czas oczekiwania na wyniki zgłoszonego zapytania jest jednym z najważniejszych czynników decydujących o ocenie systemu ekspertowego. Oznacza to, że poza poprawnie zaimplementowaną bazą wiedzy ważna jest wybrana metoda przeszukiwania. Rozróżnia się dwie metody przeszukiwania: metody ślepe (ang. *blind search*), w których algorytm nie posiada żadnych informacji na temat rozwiązania oraz metody heurystyczne, w których na podstawie informacji dodatkowych algorytm potrafi łatwiej sklasyfikować obecny stan procesu. W przypadku przeszukiwania grafów najpopularniejszymi metodami przeszukiwania są strategie [57]:

- wszerz (ang. breadth-first) – badająca węzły na tym samym poziomie, przyznając priorytety węzłom, które są zlokalizowane głębiej;
- w głąb (ang. depth-first) – w której analiza węzła zaczyna się od węzła startowego i określana jest do momentu znalezienia węzła celu;
- zachłanne (ang. *hill-climbing*) – wykorzystujące operacje ekspansji węzłów;
- z powracaniem (ang. *backtracking*) – będącą modyfikacją algorytmu w głąb. W tej wersji algorytmu wybierany jest jeden węzeł (a nie wszystkie).

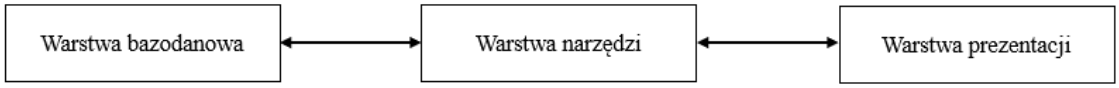
W przypadku nie spełnienia warunku węzeł ten jest dalej rozszerzany, do czasu spełnienia warunku. W sytuacji gdy węzeł nie spełni warunku wybierany jest kolejny węzeł.

W przypadku strategii heurystycznych algorytm „najpierw najlepszy" (ang. *best-first*) rozpatruje graf w oparciu o pewne informacje heurystyczne dotyczące zadania, które rozwiązuje. Wykorzystanie tych danych umożliwia skrócenie czasu przeszukiwania. Metoda „A z gwiazdką" (ang. *A-star*) wyznacza najkrótszą ścieżkę w grafie. Kolejnym przykładem strategii heurystycznej są algorytmy genetyczne, które umożliwiają analizę wielu punktów (możliwych rozwiązań) jednocześnie. W pierwszej iteracji algorytmu genetycznego losowana jest tzw. populacja osobników (zbiór węzłów), która reprezentuje prawdopodobne rozwiązania analizowanego problemu. Na podstawie oceny osobników populacji (ocena realizowana jest za pośrednictwem tzw. funkcji przystosowania) rozpoczyna się procedura selekcji (na podstawie wybranej metody). Prawdopodobieństwo wylosowania uzależnione jest od wyników funkcji przystosowania dla poszczególnych osobników. Na podstawie wybranych węzłów tworzona jest kolejna populacja. Osobniki z tzw. populacji rodzicielskiej łączą się w pary i na podstawie operacji krzyżowania powstają dwa nowe węzły. W tej sytuacji istnieje prawdopodobieństwo mutacji nowych rozwiązań, polegające na zmianie jednego genu z wybranego rozwiązania. Po zakończeniu tych operacji nowa populacja podlega analogicznej ocenie jak populacja startowa. Gdy proces oceny osobników danej populacji zakończy się, algorytm sprawdza czy cel zadania został osiągnięty – jeśli tak to algorytm kończy swoje zadanie poprzez wybranie najlepszego rozwiązania z populacji końcowej [3, 29].

Zgodnie z definicją poprzez wnioskowanie rozumie się *przedstawienie twierdzenia na podstawie analizy przesłanek* [57]. Najpopularniejsze metody wnioskowania to metoda regresywna (tzw. wstecz) polegająca na udowodnieniu hipotezy zakładając prawdziwość przesłanek, metoda progresywna (tzw. w przód) działająca w sposób odwrotny niż metoda regresywna oraz metoda mieszana wykorzystująca tzw. meta-reguły stanowiące meta-wiedzę. Na jej podstawie algorytm w sposób automatyczny wybiera metodę wnioskowania (wstecz lub w przód) [3, 57].

2.4 System nadzorowania, monitorowania i sterowania SCADA

Skomplikowana topologia systemów dystrybucji wody, zmienność parametrów pracy z zachowaniem ciągłości podejmowania szybkich i koniecznych decyzji, utrudnia zarządzanie procesem dystrybucji. Aby ułatwić ten proces nadzorcy, ważna jest implementacja nowoczesnych systemów informatycznych umożliwiających monitorowanie, nadzorowanie oraz sterowanie. Systemy tego typu powinny cechować się przejrzystością oraz intuicyjnością dlatego projektuje się je w oparciu o tzw. architekturę trójpoziomową (ang. *three layer architecture*) przedstawioną na rys. 2.4. złożoną z warstwy bazodanowej, warstwy narzędzi oraz warstwy prezentacji.

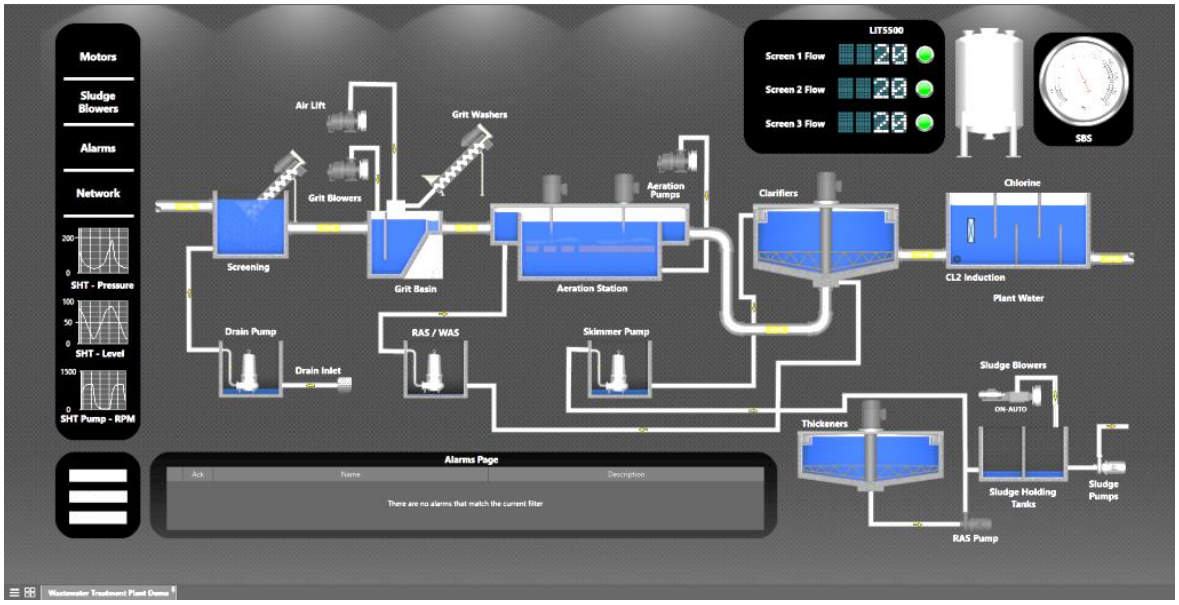


Rys. 2.4 Trójpoziomowa architektura systemów informatycznych [15]

Warstwa bazodanowa odpowiada za przechowywanie, pobieranie i zapisywanie danych. Wyniki otrzymane na podstawie odpowiednich zapytań trafiają poprzez warstwę narzędzi wprost do użytkownika. Warstwa narzędzi koordynuję pracę całego systemu, przetwarzając żądania użytkowników systemu (aplikuje reguły logiczne oraz wykonuje obliczenia). Ponadto pełni funkcję pośredniczącą między otaczającymi ją warstwami. Warstwa prezentacji jest najwyższą warstwą architektury trójpoziomowej. Jej głównym zadaniem jest tłumaczenie żądań i wyników działania systemu na język zrozumiały dla człowieka (i odwrotnie). Odbywa się to za pośrednictwem interfejsu GUI (ang. *Graphical User Interface*) lub innych urządzeń zapewniających komunikację człowiek-maszyna (np. wyświetlacz z wbudowanymi przyciskami kontrolnymi czy ekranem dotykowym). Obecnie spotyka się wiele różnorodnych systemów informatycznych służących do wizualizacji i sterowania pracą grupy obiektów. Najczęściej spotykanym oprogramowaniem jest system SCADA (ang. *Supervisory Control and Data Acquisition*), który umożliwia nadzorowanie z jednego miejsca wielu procesów jednocześnie, archiwizację dużej ilości danych procesowych oraz monitorowanie i sterowanie dalszym przebiegiem procesu. Systemy typu SCADA uważane są za

centralne (główne) narzędzia, służące do nadzorowania przebiegu procesów technologicznych. Poza nadzorem systemy te, mogą odpowiadać za [3, 15, 114]:

- monitorowanie stanu procesu i alarmowanie o stanach wyjątkowych;
- prowadzenie regulacji automatycznej (lub praca w trybie ręcznym);
- informowanie o przekroczeniu parametrów jakościowych;
- mobilne sterowanie węzłami technologicznymi;
- wybór oraz dobór wartości zadanych parametrów technologicznych;
- rejestrację i archiwizację zdarzeń;
- raportowanie i archiwizację danych;
- komunikację z warstwą sterowania bezpośredniego (np. sterownikami PLC, regulatorami czy mikrokomputerami);
- estymację zmiennych niemierzalnych;
- konfigurację ekranów synoptycznych oraz struktur algorytmicznych;
- wymianę informacji z innymi systemami lub bazami danych;
- wizualizację przebiegu procesu z wykorzystaniem ekranów synoptycznych.



Rys. 2.5 Przykładowy ekran synoptyczny systemu SCADA [115]

Stan procesu monitorowany i wizualizowany jest za pośrednictwem ekranu synoptycznego, którego przykład został przedstawiony na rys. 2.5. Dzięki takiemu rozwiązaniu w sposób łatwy i klarowny można nadzorować poszczególne etapy procesu, sprawując pieczę nad aktualnymi wielkościami pomiarowymi czy wartością zadaną sygnałów sterujących. W przypadku wystąpienia anomalii, awarii lub innej sytuacji nietypowej odpowiednie komunikaty zostają zaprezentowane w taki sposób, aby zwrócić uwagę operatora.

Monitorowanie procesu dystrybucji wody jest elementem niezbędnym do zarządzania jego eksploatacją. Należy również pamiętać, że zgodnie z przepisami polskiego prawa [71, 79] oraz normą EN-PN 805:2002 jest to wręcz obowiązkowe. Celem powyższego zadania *jest minimalizacja zakłóceń w procesie zaopatrzenia w wodę oraz minimalizacja wpływu na środowisko i zdrowie publiczne. Sieci wodociągowe powinny być stale monitorowane i nadzorowane pod kątem pomiarów: przepływu wody, ciśnienia wody, jakości wody oraz poziomu świadczonych usług* [15, 49].

2.5 Integracja złożonych systemów informatycznych

Mianem systemu określa się obiekt lub zjawisko stanowiące fragment analizowanej rzeczywistości stanowiący pewnego rodzaju całość [15, 28, 60]. Zlokalizowany jest w otoczeniu, z którym zachodzą różne interakcje. Zależności między tymi bytami są ograniczone, przez co można uznać system za autonomiczny. Ze względu na autonomiczność systemów konieczna staje się ich integracja. W tym celu wykorzystuje się już dostępne systemy integracyjne, tworzy nowe bądź aktualizuje się je, dodając dodatkową funkcjonalność, tzw. API (ang. *Application Programming Interface*), które jest określonym zbiorem reguł oraz opisów, umożliwiających komunikację różnych systemów ze sobą. Przykładowymi systemami wymagającymi integracji mogą być: systemy SCADA z systemem bazy danych (przechowujące informację historyczne o przebiegu procesu), baza danych zawierająca zbiór informacji pozwalających na generowanie modeli hydraulicznych systemów dystrybucji wody (w celu analizy scenariuszy testowych, np. sprawdzenie czy obecna sieć wodociągowa zaopatrzy nowo powstałe osiedle w wystarczającą ilość wody zdatnej do spożycia) czy inne rodzaje bazy danych (w tym bazy wiedzy niezbędne do działania systemów wspomagania decyzji czy systemów ekspertowych). Należy pamiętać, że stopień realizacji poszczególnych

systemów może się różnić. Wyróżnia się cztery etapy (poziomy) rozwoju oprogramowania (modelu) [8]:

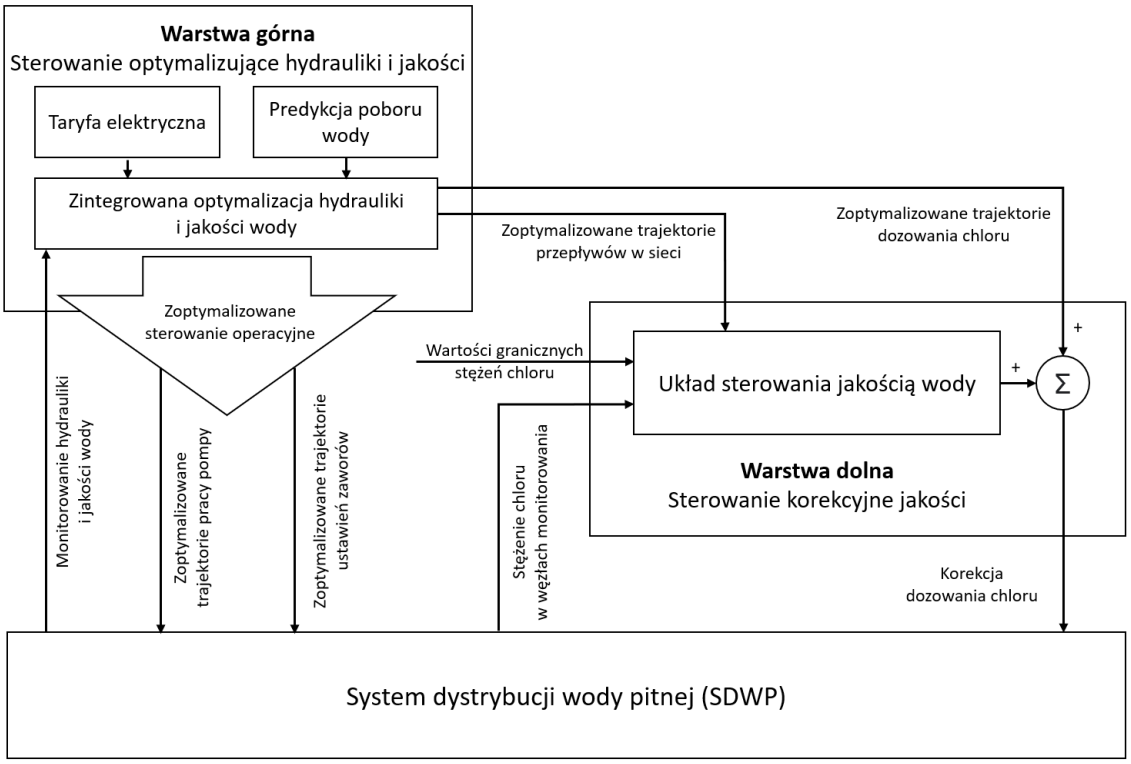
- poziom pierwszy – modele realizowane w celach naukowo-dydaktycznych. Rozwijane są przez studentów oraz naukowców w celu analizy niewielkich pod względem skali i zasięgu przypadków badawczych;
- poziom drugi – modele rozwijane przez grupy naukowców, realizujące i rozwiązujące złożone pod względem skali i zasięgu projekty naukowe;
- poziom trzeci – modele, które zostały wielokrotnie przetestowane przez specjalnie opracowane i zaimplementowane testy. Zajmują się rozwiązywaniem praktycznych problemów naukowych. Za pośrednictwem GUI możliwa jest prosta i przejrzysta wymiana informacji, wprowadzanie danych wejściowych, wyświetlanie wyników itp. Obsługa systemu realizowana jest z wykorzystaniem instrukcji, opisującej sposób działania systemu (w tym wymagania techniczne);
- poziom czwarty – model, który nie podlega już modyfikacji ze strony twórcy oprogramowania. Tego typu systemy wykorzystywane są przez użytkowników końcowych (operatorów i nadzorców procesu). Komunikacja eksploatatora z systemem odbywa się na podstawie czarnej skrzynki (użytkownik nie posiada wiedzy na temat sposobu działania systemu) tzn. wysyła dane wejściowe, oczekując na dane wyjściowe.

2.6 Przykłady zintegrowanych systemów informatycznych w inżynierii środowiska

Zjawisko integracji wyizolowanych systemów informatycznych pochodzących od wielu różnych dostawców, analizujące dane z różnych warstw struktury organizacyjnej danego przedsiębiorstwa intensyfikuje się. Powodem wzrostu zainteresowania tego typu rozwiązaniami jest przede wszystkim presja rynku oraz potrzeba rozwoju przedsiębiorstwa. W erze dynamicznego rozwoju technologii coraz częściej można spotkać się więc z propozycjami złożonych systemów integrujących, wykorzystujących nowoczesne rozwiązania technologicznie w zakresie analizy danych i wspomagania decyzji. W przypadku przedsiębiorstw wodociągowych najczęściej wykorzystywanymi

systemami informatycznymi są systemy ERP, SCADA oraz GIS (*ang. Geographic Information System)*, których przykładową strukturę integracji zaproponowano w pracy [18]. Przedstawione rozwiązanie w oparciu o techniki modelowania analitycznego i numerycznego sieci wodociągowych oraz system informacji geograficznej rozpowszechnia istotne informacje wśród zespołów naprawczych. Stan elementów infrastruktury wodociągowej analizowany jest na podstawie modelowania matematycznego, sztucznych sieci neuronowych oraz danych historycznych. Dzięki zaproponowanemu rozwiązaniu możliwa jest wizualna reprezentacja stanu sieci w systemie GIS oraz szersza analiza wiedzy zdobytej podczas naprawy w celu utrzymania sprawności systemu dystrybucji wody na odpowiednim poziomie. Przykład koncepcji zintegrowanego zarządzania systemami wodociągowymi opisano również na przykładzie doświadczeń australijskich przedsiębiorstw w pracy [56]. Wyzwania, kierunek badań oraz przykłady rozważające sposoby integracji systemów IT w inżynierii środowiska przedstawiono również w pracach [22, 25, 26, 44, 55, 56, 94].

Warto nadmienić, że poza integracją systemów działających w różnych warstwach struktury organizacyjnej przedsiębiorstwa istotna jest także współpraca systemów działających w obrębie jednej warstwy. W przypadku systemów dystrybucji wody uwzględniając przykładowo powiązanie między jakością wody a hydrauliką, system zarządzania powinien być wspierany przez odpowiednio zintegrowany system sterowania. W pracy [52] przedstawiono system sterowania w postaci dwuwarstwowej odpowiedzialny za minimalizację kosztów prowadzenia procesu dystrybucji w oparciu o aspekt hydrauliczny oraz jakościowy. Na podstawie informacji z systemu monitorowania jakości i systemu monitorowania hydrauliki zadaniem zaproponowanego rozwiązania jest wygenerowanie w sposób efektywny odpowiednich wartości sterowania. Podstawowym zadaniem systemu odpowiedzialnego za monitorowanie jakości SZZwW jest dostarczenie informacji o stanie jakości wody, w tym wartości stężeń chloru w poszczególnych węzłach sieci oraz zbiornikach wodnych. System monitorowania hydrauliki natomiast przekazuje informacje o wartościach takich jak: objętości wody przechowywanych w zbiornikach, natężeniach przepływu oraz ciśnieniach [52]. Dwuwarstwowa struktura systemu sterowania siecią dystrybucji wody przedstawiona została na rys. 2.6.



Rys. 2.6 Dwuwarstwowa struktura sterowania siecią dystrybucji wody [52]

Obecnie, na rynku można zaobserwować również wiele firm proponujących gotowe rozwiązania z dziedziny integracji systemów IT w branży wodno-kanalizacyjnej. Przykładem może być produkt TP-AQUA firmy Logical Poland Sp. z o.o., który umożliwia zarządzanie całym przedsiębiorstwem wod - kan. System ten składa się z czterech głównych modułów odpowiedzialnych za: system informacyjny kierownictwa, system obsługi dokumentów, system zarządzania infrastrukturą techniczną (w tym: GIS, SCADA, audyty strat wody oraz modele hydrauliczne sieci wodociągowych) oraz system zarządzania w obszarze ekonomicznym (w tym: biuro obsługi klienta, moduł ERP odpowiedzialny za finanse i księgowość). Za integrację powyższych modułów odpowiada wspólna baza danych ułatwiająca prostą wymianę danych między poszczególnymi pionami przedsiębiorstwa [123]. Z zagranicznych firm możemy wyróżnić natomiast EDAMS Technology. Firma oferuje produkt o nazwie Water and Santation Utilities, charakteryzujący się modułowością, pozwalający na skalowalność wdrożenia dostosowanego do wielkości przedsiębiorstwa. System ten ma wbudowany system GIS, który może zostać połączony z takimi systemami jak: ArcGIS, QGIS czy

SM GIS. Ponadto system pozwala na integracje na poziomie baz danych, procesów biznesowych, transakcji (unikając w ten sposób zjawiska powielania danych), ale również obejmuje podstawowe funkcje biznesowe przedsiębiorstwa, zarządzania operacyjnego, planowania i zarządzania utrzymaniem ruchu [124].

Ze względu na wzrost złożoności techniczno-technologicznej przedsiębiorstwa w celu efektywnego zarządzania coraz częściej poza integracją wyizolowanych systemów informatycznych oczekuje się wdrożenia modułów odpowiedzialnych za wspomaganie decyzji. Obecnie wdrożone tego typu systemy odpowiadają za archiwizację danych dotyczących zleceń naprawczych oraz budowanie modeli decyzyjnych na podstawie określonych wskaźników eksploatacyjnych. Następnie w oparciu o algorytmy interpretacji uzyskanych wartości, zostają one poddane analizie na podstawie ograniczonego zestawu wyznaczonych miar. W pracy [105] zaproponowano następujące podejścia:

- ISOZE (Inteligentny System Obsługi Polityki Eksploatacyjnej), który umożliwia ocenę polityki eksploatacyjnej, przedstawionej za pośrednictwem rozbudowanego zbioru miar (30 wskaźników), których wartości obliczane są na podstawie danych zgromadzonych w systemie ERP;
- SMOPE (Scenariuszowy Moduł Opisu Polityki Eksploatacyjnej), który na podstawie taksonomicznej agregacji wartości kluczowych cech wynikających z prac naprawczych pozwala wyznaczyć ocenę polityki eksploatacyjnej.

Niestety mając na uwadze indywidualne i zróżnicowane potrzeby konkretnych przedsiębiorstw zaopatrzenia w wodę nie możliwym staje się porównanie tego typu systemów pod względem modelu, struktury czy funkcjonowania [105].

3. Charakterystyka sieci wodociągowych

3.1 Wprowadzenie

System zbiorowego zaopatrzenia w wodę (SZZwW) jest zbiorem elementów wykonawczych oraz obiektów technicznych współpracujących ze sobą. Jego zadaniem jest dostarczenie produktu (wody zdatnej do spożycia) do wskazanych odbiorców (mieszkańców miasta, przemysłu, itp.) [15, 19, 72]. Woda, która trafia do odbiorców powinna spełniać wymogi określone przez ustawy i rozporządzenia wydawane przez organy państwowe [78, 96]. W systemie zbiorowego zaopatrzenia w wodę przede wszystkim należy wyróżnić: ujęcie wody, stację uzdatniania wody (SUW) oraz sieć wodociągową. Ujęcie wody wyposażone jest w urządzenia i infrastrukturę budowlaną odpowiedzialną za pobranie wody ze źródła oraz dostarczenie jej do SUW. Ponadto posiada niezbędne urządzenia techniczno-technologiczne konieczne do dostosowania jakości wody surowej do jakości wymaganej przez odpowiednie rozporządzenia. Woda pobrana przez pompy głębinowe (wstępnie przefiltrowana za pośrednictwem gruntu) wykorzystując układ połączonych ze sobą przewodów wodociągowych trafia do filtrów. Należy zwrócić uwagę, że woda ta na ogół nie zawiera zanieczyszczeń mechanicznych, jednakże bardzo często w skład tego półproduktu wchodzą związki żelaza i manganu, agresywny dwutlenek węgla oraz niewielka ilość tlenu [5, 81]. Stacje uzdatniania wody nadzorują proces uzdatniania, który polega na filtracji oraz napowietrzaniu. Podstawowymi elementami wchodzącymi w skład SUW są: filtry, aeratory, agregaty prądotwórcze, zbiorniki wody czystej oraz inne urządzenia techniczne, których zadaniem jest dostosować jakość wody ujmowanej do jakości wymaganej przez odbiorców. Filtr to zamknięty zbiornik (najczęściej wykonany ze stali) w kształcie walca, którego pole przekroju poprzecznego na dnie oraz szczycie zmniejsza się w sposób liniowy tworząc zaokrąglenie. Funkcję materiału filtracyjnego (złoża) w tego typu zbiornikach pełnią kamienie kwarcowe o różnej gradacji, które ułożone są od najmniejszej średnicy na dole zbiornika po największą u góry. Stan złoża pełni kluczową rolę w procesie filtracji. Od jego objętości oraz czystości uzależniona jest jakość uzdatnianej wody. Złoże kwarcowe posiada wysokie właściwości utleniające oraz adsorpcyjne. Oznacza to, że doskonale sorbuje ono związki żelaza i manganu oraz katalizuje utlenianie żelaza i manganu [5, 43, 81]. Woda czysta (uzdatniona) jest odprowadzana z filtra za pośrednictwem przewodu

wodociągowego do zbiornika wody czystej. Cały proces realizowany jest pod ciśnieniem, dlatego ważne jest ciągłe monitorowanie jego wartości. W przypadku wzrostu ciśnienia lub pogarszającej się jakości uzdatnianej wody konieczne jest przeprowadzenie płukania filtra lub wymiany jego złoża. Proces płukania filtra odbywa się około raz na trzy/cztery dni i jest w głównej mierze uzależniony od decyzji eksploatatora SUW [5, 43, 84]. Warto również wspomnieć, że dwa razy w miesiącu z każdego filtra i zbiornika wody czystej pobiera się próbki, w celu kontroli jakości wody. Na jej podstawie, można określić jakość oczyszczania wody oraz stopień zabrudzenia filtrów.

3.2 Sieci wodociągowe – przebieg procesu dystrybucji wody

Przełomem w dziedzinie transportu wody na dalekie odległości było wybudowanie przez Rzymian w II w. n.e. akweduktów. Spływ wody w odpowiednim kierunku umożliwiało wykorzystanie siły grawitacji oraz odpowiedniego kąta nachylenia całej konstrukcji. Większa część rzymskich wodociągów przebiegała pod ziemią (przeważały przewody wykonane z ołowiu lub terakoty), natomiast część nadziemna była zazwyczaj umieszczana w dwu lub trzykondygnacyjnej konstrukcji z cegieł. Układ podziemnych przewodów pełnił przede wszystkim funkcję obronną zasobów wodnych. Jakość wody źródła, z którego była natomiast czerpana, badana była na podstawie obserwacji zwierząt, zaspokajających pragnienie z analizowanego źródła [19]. Aktualnie transport oraz dystrybucja wody odbywa się za pośrednictwem złożonych systemów wodociągowych, które wg. definicji składają się z zespołu urządzeń inżynierskich oraz obiektów technicznych [19, 33, 72]. Oznacza to, że określenie „wodociągi" wykorzystuje się w szerszym znaczeniu niż tylko użycia przewodów do transportu wody [72]. O ile sposoby projektowania, budowania czy wykorzystywane materiały ulegały zmianie na przestrzeni lat, o tyle przeznaczenie całej konstrukcji pozostało niezmienne. Do dnia dzisiejszego sieci wodociągowe mają za zadanie dostarczać w sposób niezawodny wodę do odbiorców w taki sposób, aby zaspokoić podstawowe zapotrzebowanie na cele bytowo-gospodarcze oraz przeciwpożarowe [7].

3.2.1 Budowa sieci wodociągowych

Zgodnie z przepisami polskiego prawa podczas budowy nowego odcinka sieci wodociągowej, modernizacji sieci lub wystąpienia awarii i konieczności wymiany

przewodu należy stosować materiały oraz średnice przewodów, które zapewnią efektywną pracę całej sieci przy zachowaniu minimalnej straty energii. W przypadku materiałów potrzebnych do realizacji sieci wodociągowych wszystkie muszę spełniać wymogi opisane w ustawie o wyrobach budowlanych. Materiały te muszą posiadać: znak CE (informujący o zgodności materiału z europejską aprobatą techniczną lub krajową specyfikacją techniczną państwa członkowskiego Unii Europejskiej) oraz atest higieniczny Państwowego Zakładu Higieny (PZH). W przypadku braku znaku CE, materiał powinien posiadać charakterystyczny znak budowlany (opisany w art. 5 ust. 1. pkt 3 ww. ustawy). Ponadto wszystkie wykorzystywane materiały powinny posiadać określone w normach właściwości mechaniczne oraz być dobierane w taki sposób aby skład chemiczny oraz oddziaływanie z resztą konstrukcji nie skutkowało pogorszeniem zdatności do spożycia transportowanej wody [96]. W trakcie budowy sieci wodociągowych należy pamiętać, aby przykrycie przewodów było nie mniejsze niż półtora metra (w przypadku rur PE – 1.7 metra). Nad przewodami należy umieścić tzw. drut sygnalizacyjny (miedziany DY 1.0 mm^2), pozwalający na wyznaczenie ewentualnej trasy sieci (za pośrednictwem specjalistycznego sprzętu pomiarowego). Ponadto 30 centymetrów nad przewodem należy umieścić taśmy ostrzegawcze (w kolorze niebieskim), których zadaniem jest ochrona przed mechanicznymi uszkodzeniami (np. w trakcie robót ziemnych).

Podstawowym elementem systemu dystrybucji wody są przewody. Ze względu na rodzaj materiału, z jakiego zostały wykonane możemy wyróżnić: przewody z tworzyw sztucznych oraz przewody metalowe. W zależności od lokalizacji umiejscowienia przewodu wyróżnić można między innymi [7]:

- **przewody typu PE** – charakteryzują się niewielkim ciężarem właściwym (w porównaniu z innymi typami przewodów takimi jak PVC, stal czy żelazo). Łączenie rur odbywa się poprzez wykorzystanie zgrzewania typu doczołowego lub elektrooporowego (w przypadku węzłów przewody łączy się za pośrednictwem połączeń kołnierzowych). Przewody tego typu cechuje wysoka odporność na korozje naprężeniową oraz długotrwałe ciśnienie hydrauliczne (bardzo wysoki stopień bezawaryjności). W przypadku sieci wodociągowych rury powinny zostać wykonane

z materiału PE-100, w których ciśnienie robocze nie może być niższe niż 1.0 MPa (zgodnie z normą PN-EN 12201 [64]);

- **przewody z żeliwa sferoidalnego** – charakteryzują się wysoką odpornością na zgniatanie oraz ściskanie. Dzięki zastosowaniu specjalnej powłoki zewnętrznej przewody tego typu cechuje wysoka odporność na korozje. Wraz ze wzrostem średnicy nominalnej przewodu grubość ścianki również ulega zwiększeniu. W przypadku tego typu przewodów stosuje się połączenia kielichowe, kołnierzowe w punktach węzłowych. Dopuszcza się również stosowanie w połączeniach węzłowych trójników kielichowo-kołnierzowych;
- **przewody stalowe** – posiadają dobre właściwości mechaniczne oraz są wytrzymałe na ściskanie, rozciąganie oraz zginanie. Zgodnie z obecnymi przepisami należy stosować rury stalowe ze szwem spiralnym. Ze względu na konieczność stosowania zewnętrznych powłok ochronny (chroniących przed korozją) – koszt tego typu przewodów znacznie wzrasta. Przewody stalowe łączy się poprzez spawanie, natomiast w przypadku węzłów wykorzystuje się kołnierze;
- **przewody PVC** – wykorzystuje się najczęściej w sytuacji unifikacji materiału przewodów wodociągowych (z zastrzeżeniem, że nowy odcinek nie będzie narażony na intensywne obciążenia dynamiczne). Używane materiały powinny być wykonane z materiału jednorodnego (zgodnego z normami: PN-EN 1452-2 oraz PN-EN 1452-3 [65, 66]). W przypadku łączeń wykorzystuje się standard kielichowy (wraz z uszczelką z atestem PZH). Natomiast w przypadku łączenia przewodu z węzłem korzysta się ze standardu kołnierzowego (ciśnienie robocze 1.0 MPa).

Wyżej wymienione elementy są fundamentem każdego systemu wodociągowego, stanowiąc nośnik, używany do transportowania wody uzdatnionej za pośrednictwem stacji uzdatniania wody. Odgałęzienia wodociągu wykonuje się poprzez wprowadzenie do sieci (najczęściej poprzez wcięcie) trójnika lub czwórnika wykonanego z żeliwa sferoidalnego.

Kolejnymi ważnymi elementami wchodzącymi w skład sieci wodociągowej są zasuwy oraz przepustnice. Przepustnica odpowiada za regulację ilości wody doprowadzanej do wybranego odcinka sieci, natomiast zasuwa odpowiada za zamknięcie (odizolowanie) wybranego odcinka sieci wodociągowej. W przypadku przepustnic występujących w przewodach, w których średnica nominalna jest większa niż 500 milimetrów konieczne jest zastosowanie dodatkowego tzw. by-passu, który powinien być zintegrowany z korpusem oraz spełniać zasadę modułowości (łatwy sposób demontażu w celu wymiany lub konserwacji) [7]. W przypadku zasuw, wykorzystuje się takie, których ciśnienie nominalne jest nie mniejsze niż 1.0 MPa oraz uszczelnienie jest typu miękkiego. Wymiary zasuw muszą być zgodne z normą PN-EN 1092-2 [67]. Warto dodać, że zasuwy pełnią również rolę ochronną i regulacyjną. W przypadku wystąpienia awarii lub skażenia umożliwiają one odseparowanie wrażliwego odcinka od reszty systemu. Ponadto za pośrednictwem regulacji stopnia zamknięcia zasuwy możliwa jest zmiana prędkości transportowanej wody. Dokumentacja techniczna zasuw informuje przede wszystkim nadzorcę stanu procesu dystrybucji wody o ciśnieniu roboczym, oporach miejscowych oraz współczynniku przepływu [24].

W celu utrzymania odpowiedniego ciśnienia wewnątrz systemu dystrybucji wody, w newralgicznych punktach sieci wodociągowej buduje się tzw. stacje podnoszenia ciśnienia (inaczej: pompownie). W skład takiej stacji wchodzi zespół obiektów budowlanych oraz pomp, które są połączone z silnikami napędowymi za pośrednictwem sprzęgła. Rodzaj stosowanych silników jest uzależniony od funkcjonalności danej stacji podnoszenia ciśnienia. W przypadku pompowni bez dodatkowej funkcji wykorzystuje się wyłącznie silniki elektryczne, natomiast w pompowniach przeciwpożarowych stosuje się silniki elektryczne oraz spalinowe na wypadek braku zasilania. Głównym zadaniem tego typu stacji jest podnoszenie wody z poziomu niżej położonego na poziom położony wyżej lub przetłoczenie jej do obszaru o wyższym ciśnieniu [19]. W sieciach wodociągowych oraz stacjach podnoszenia ciśnienia stosuje się [19, 20]:

- **pompy głębinowe** – służące najczęściej do poboru wody ze studni w celu zasilenia gospodarstw domowych lub dostarczenia wody do stacji uzdatniania wody. Cechują się niewielką średnicą co pozwala na dostęp do

studni o małym polu powierzchni przekroju poprzecznego. Dodatkowym atutem jest możliwość pracy urządzenia pod powierzchnią wody;

- **pompy wirowe** (z wałem pionowym lub poziomym) – w których za pośrednictwem wirnika (w kształcie łopatek) zwiększany jest moment pędu cieczy. Zaletą tego typu pompy jest duża wydajność (przy stosunkowo niewielkiej wysokości podnoszenia) oraz równomierność ruchu przy ustalonych warunkach pracy. Minusem natomiast jest brak możliwości samo zassania oraz wrażliwość na zanieczyszczenia występujące w pompowanej cieczy [41].

Poprzez efektywność pracy pompy rozumie się jej zdolność podnoszenia (tzn. wysokość w metrach na jaką dana pompa jest w stanie efektywnie wynieść ciecz). Aby opisać charakterystykę pracy takiej pompy wykorzystuje się stosunek podnoszenia pompy do natężenia przepływu cieczy przez tą pompę (zakładając w przypadku pomp wirowych stałą prędkość obrotową wirnika). W celu wyznaczenia rzeczywistego punktu pracy należy porównać dokumentację techniczną oraz charakterystykę analizowanej pompy z dokumentacją wykorzystywanego przewodu. Na podstawie tych danych można np. określić zasięg analizowanego typoszeregu pomp [20, 109].

Istnieje wiele sposobów umożliwiających zmianę parametrów pracy wykorzystywanych pomp. Podstawową, a zarazem najmniej ekonomiczną metodą zmiany wydajności pompy, jest dławienie. Dławienie polega na zmniejszaniu się ciśnienia cieczy, przy przepływie przez przeszkodę w postaci zwężonego przekroju [86]. Kolejną metodą regulacji pracy pompy jest zmiana prędkości obrotowej za pośrednictwem przetwornicy częstotliwości. Ze względu na brak powstawania strat związanych z dławieniem jest ona przeciwieństwem (pod względem ekonomicznym) metody opisanej powyżej. Metoda upustowa odprowadza częściowo wodę znajdującą się w przewodzie tocznym do przewodu ssawnego znajdującego się przed pompą [19, 20]. Zmniejszenie parametrów pracy pompy lub zespołu pomp odbywa się również poprzez zmianę geometrii wirnika, polegającej na zmianie średnicy zewnętrznej wirnika [19].

Zbiorniki są kolejnym istotnym elementem systemów zaopatrzenia w wodę. W zależności od ich lokalizacji mogą nie tylko magazynować wodę surową lub już

uzdatnioną, ale również pełnić funkcje wyrównawcze. Tego typu zbiorniki bilansują nierównomierności wynikające między dostawą wody w sieci wodociągowej, a jej poborem, co skutkuje zwiększeniem niezawodności takiej sieci. Poza zbiornikami wyrównawczymi i magazynującymi istnieją jeszcze zbiorniki przeciwpożarowe, których zadaniem jest magazynowanie wody na cele p.poż. Pojemność zbiornika wyliczana jest na podstawie dobowego bilansu wodnego. Dobowy bilans wodny zbiornika obliczany jest na podstawie objętości wody dopływającej i wypływającej ze zbiornika. W oparciu o dane historyczne związane z bilansem analizowanego zbiornika określane jest średnie zapotrzebowanie na wodę we wskazanym obszarze. Wybierając pojemność nowego zbiornika, pod uwagę należy wziąć również [19, 111]: wydajność wodociągu, lokalizację zbiornika, konstrukcję i kształt zbiornika, nierównomierności rozbioru wody w sieci wodociągowej oraz wymagania co do czasu przechowywania wody w zbiorniku.

Poza dostarczeniem wody do odbiorców sieć wodociągowa pełni również funkcję transportu wody do celów przeciwpożarowych. W tym celu wykorzystuje się hydranty tj. urządzenia pozwalające na bezpośredni pobór wody z sieci wodociągowej. Budowa hydrantu (zawór oraz złącze) pozwala na szybkie podłączenie się wozu strażackiego do sieci oraz otwarcie zaworu umożliwiającego rozpoczęcie akcji gaśniczej. Ze względu na lokalizację hydrantów rozróżniamy hydranty zewnętrzne (tzw. uliczne): nadziemne i podziemne oraz wewnętrzne (budynkowe). Sieć dystrybucji wody wyposażona jest jeszcze w wiele różnych elementów, takich jak: zawory odpowietrzające czy anty-skażeniowe. Ich zadaniem jest kolejno usunięcie powietrza z układów hydraulicznych oraz ochrona sieci przed rozpowszechnianiem występującego skażenia w sieci.

3.2.2 Rodzaje sieci wodociągowych

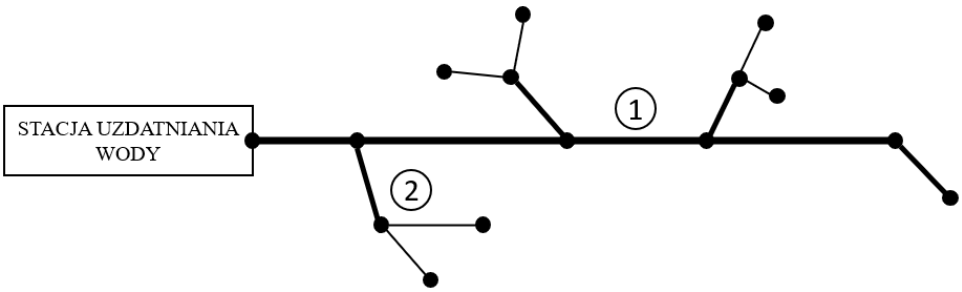
Ze względu na wzrost świadomości o procesach zachodzących w przyrodzie oraz wpływu jakości wody na życie człowieka każda sieć wodociągowa powinna spełniać określone wymagania. Przede wszystkim systemy dystrybucji wody powinny [112]:

- dostarczyć wodę o pożądanym ciśnieniu;
- spełniać zapotrzebowanie odbiorców objętych działaniem sieci;
- zagwarantować wysoką niezawodność;

- zapewnić możliwie najtańszy koszt budowy i eksploatacji.

Wymagania wymienione powyżej stanowią bazę podczas projektowania tego typu systemów. Odpowiedni dobór układu sieci przewodów, właściwy wybór materiałów spełniających wymagania ustawowe, średnice przewodów (uzależnione od liczby ludności i objętości transportowanej wody) odgrywają kluczową rolę w celu ich spełnienia. Ze względu na różną konfigurację przewodów, wyróżnić można trzy rodzaje sieci wodociągowych [19, 112]:

- **sieć rozgałęzieniowa (promienista)** – zbudowana jest z jednego przewodu głównego o największej średnicy (tzw. magistrala), do którego podłączone są przewody o coraz mniejszych średnicach. Tego typu rodzaj sieci charakteryzuje się niską niezawodnością – jest to sieć podatna na awarię. Spowodowane jest to faktem zasilania sieci tylko z jednej strony co w przypadku godziny największego rozbioru może skutkować obniżeniem ciśnienia w całej sieci. W przypadku wystąpienia awarii na sieci wodociągowej, często konieczne jest wstrzymanie dostawy wody dla wszystkich odbiorców objętych działaniem sieci [112]. Przykład topologii sieci wodociągowej tego typu przedstawiony został na rys. 3.1.

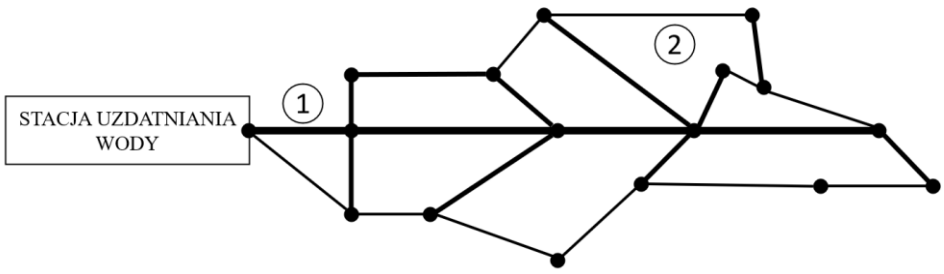


Rys. 3.1 Topologia sieci rozgałęzieniowej [19, 112]

Numerem (1) oznaczono przewód główny (magistralę), odpowiedzialny za rozdystrybuowanie wody do poszczególnych dzielnic miasta czy zakładów przemysłowych. Ponadto magistrala ta dostarcza również wodę do przewodów o mniejszej średnicy. Za przewody główne zgodnie z obowiązującymi normami uważa się przewody, których średnica

przekracza 300 milimetrów. Przewody oznaczone numerem (2) nazywane są przewodami rozdzielczymi. Przewody tego typu charakteryzują się różną średnicą (zazwyczaj większą niż 100 milimetrów i mniejszą niż 300 milimetrów, w zależności od zapotrzebowania). Do przewodów rozdzielczych bezpośrednio pozostają podłączani odbiorcy, za pośrednictwem tzw. przyłączy, w celu czerpania wody z sieci wodociągowej. Topologia tego typu używana jest dotychczas, najczęściej w sieciach tymczasowych oraz na obszarach wiejskich (ze względu na niskie koszty eksploatacji i małe wymagania).

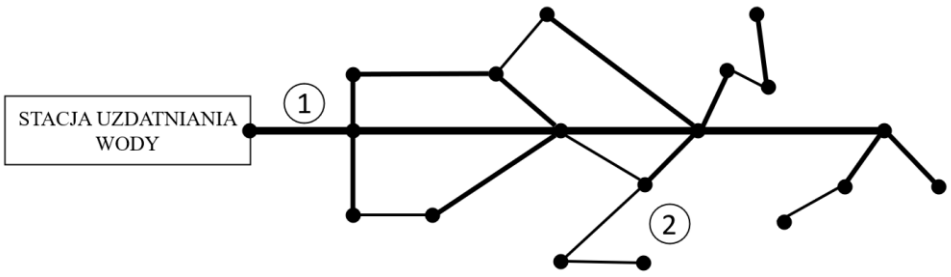
- **sieć pierścieniowa (obwodowa) –** charakteryzuje się znacznie większą niezawodnością niż sieć promienista. Ze względu na jej strukturę dostarczenie wody do jednego z węzłów może odbywać się na wiele sposobów. W przypadku tej topologii istotne jest również poprawne rozmieszczenie zasuw, ponieważ w sytuacji wystąpienia awarii na jednym z przewodów odpowiednie ich rozmieszczenie może spowodować odizolowanie mniejszego fragmentu sieci w celu jej naprawy (a nie jak w sytuacji topologii promienistej odizolowanie całej sieci). Tego typu rozwiązanie umożliwia więc dostarczenie wody z kilku różnych kierunków. Dodatkowym atutem tego typu topologii, jest niewielka zmiana ciśnienia na obszarze całej sieci (w przeciwieństwie do wcześniej omawianej struktury). Cecha ta ma wysoki wpływ na niezawodność procesów przeciw-pożarowych, ponieważ zmiany ciśnienia spowodowane większym rozbiorem wody nie mają istotnego wpływu na procesy gaśnicze. Przykład topologii sieci obwodowej został przedstawiony na rys. 3.2.



Rys. 3.2 Topologia sieci obwodowej [19, 112]

Na rys. 3.2 numerem (1) oznaczono przewód magistralny, natomiast numerem (2) przewód rozdzielczy. Zadania poszczególnych przewodów zostały opisane na etapie opisu rys. 3.1 sieci rozgałęzieniowej. Systemy wodociągowe zaprojektowane i wykonane w strukturze obwodowej znacznie lepiej znoszą uderzenia hydrauliczne, które powstają w momencie nagłego zatrzymania przepływu wody [19]. Pomimo wielu zalet tego rodzaju sieci, znaczącą wadą jest koszt budowy oraz eksploatacji. Woda docierająca do określonych obszarów miast transportowana jest za pośrednictwem wielu przewodów co oznacza zwiększenie kosztów budowlanych.

- **sieć mieszana** – jest hybrydą topologii magistrali oraz topologii pierścieniowej, co sprawia, że jest obecnie najczęściej spotykanym rodzajem sieci wodociągowej. Za pośrednictwem topologii rozgałęzieniowej woda transportowana jest do oddalonych jednostek osadniczych (np. wsi) czy zakładów przemysłowych, natomiast topologia obwodowa zasila centrum miasta i pobliskie dzielnice. W przypadku rozpatrywania niezawodności sieci odbiorcy oddaleni od centrum są bardziej narażeni na przerwy w dostawie wody niż ci z centrum. Przykład topologii sieci mieszanej został przedstawiony na rys. 3.3.



Rys. 3.3 Topologia sieci mieszanej [19, 112]

Na rys. 3.3 numerem (1) oznaczono przewód magistralny, natomiast numerem (2) przewód rozdzielczy. Zadania poszczególnych przewodów zostały opisane na etapie opisu rys. 3.1 sieci rozgałęzieniowej.

Wybór odpowiedniej topologii uzależniony jest przede wszystkim od lokalizacji obszaru, który ma zostać zaopatrzony w wodę zdatną do spożycia. Obszary wiejskie o niskim stopniu zaludnienia zasilane są zazwyczaj z systemu rozgałęzieniowego. Spowodowane jest to minimalizacją kosztów budowy i eksploatacji. Ponadto rzadko zdarza się sytuacja, w której na obszarze wiejskim znajduje się przedsiębiorstwo przemysłowe, które musi mieć stały dostęp do zasobów wody czystej. Obszary o gęstym zaludnieniu wykorzystują układ obwodowy, zapewniający sobie tym samym system dystrybucji wody o dużej niezawodności.

3.2.3 Sposoby wykrywania i naprawy awarii

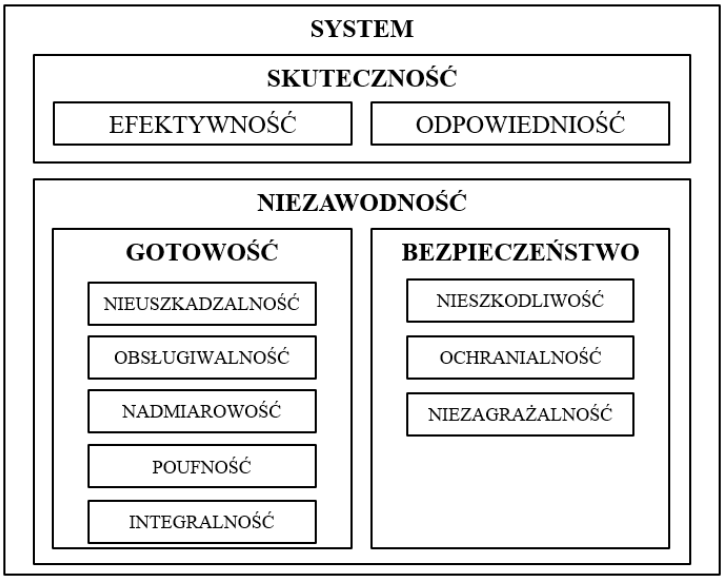
Miejsce wypływu lub gromadzenia się wody nie zawsze jednoznacznie wskazuje miejsce wystąpienia awarii. Bardzo często bywa tak, że woda wypływa na powierzchnię w odległości kilku do kilkunastu metrów od faktycznego miejsca uszkodzenia przewodu (spowodowane jest sposobem zagospodarowania nawierzchni np. jej utwardzeniem przez asfalt, beton czy kostkę brukową). W przypadku gdy czas trwania awarii jest długi (czas trwania wyliczany jest na podstawie różnicy czasu wykrycia i czasu powstania usterki) prowadzenie robót naprawczych powinno odbywać się z użyciem pompy odpompowującej nadmiar wody z wykopu. Brak tej czynności może spowodować trwałą zmianę nawierzchni w pobliżu awarii co może skutkować jej zapadnięciem [101]. Lokalizacja obszaru występowania zmian w podłożu nawierzchni jest dość trudnym zagadnieniem. Istnieje jednak wiele sposobów wykrywania wycieków i lokalizacji awarii. Przykładem koncepcji systemu detekcji awarii może być propozycja systemu opisanego w artykułach [73, 74, 75, 76, 103, 108], który na podstawie informacji o przepływach wody z punktów monitorowania zainstalowanych w systemie dystrybucji wody oraz sztucznej sieci neuronowej, wykrywa, lokalizuje i powiadamia zarządcę sieci o prawdopodobieństwie wystąpienia wycieku na wskazanym obszarze. W artykułach tych porównano różne rodzaje sztucznych sieci neuronowych i ich skuteczność w zależności od analizowanej topologii sieci wodociągowej. Badania te jednoznacznie udowodniły skuteczność wykorzystania algorytmów heurystycznych w rozwiązywaniu tego typu problemów. Kolejnym przykładem wspomagającym proces wykrywania i minimalizacji liczby awarii jest wdrożenie systemu monitorowania, które zostało opisane w pracach [52, 63]. Odpowiednia alokacja punktów pomiarowych umożliwia

analizę danych i zmniejszenie wskaźnika awaryjności. Innym przykładem systemu detekcji awarii jest wykorzystanie analizy skupień oraz zbiorów rozmytych do rozpoznawania wzorców [106]. Dzięki wykorzystaniu narzędzi do symulacji przebiegu procesu dystrybucji wody, możliwe jest wykrycie wzorców i wskazanie z wysokim współczynnikiem prawdopodobieństwa miejsca wystąpienia awarii.

Ostatnio zauważyć można również wzrost zainteresowania algorytmami ewolucyjnymi i wykorzystaniem ich w detekcji anomalii występujących na sieci wodociągowej [1, 9, 54]. Na podstawie modelu hydraulicznego sieci wodociągowej i implementacji algorytmów genetycznych możliwa jest równoległa symulacja wielu sytuacji awaryjnych na sieci wodociągowej. W zależności od analizowanego problemu (np. wykrywanie wycieków), algorytmy te potrafią z wysokim prawdopodobieństwem wskazać miejsce ich wystąpienia. Według [101] roboty naprawcze sieci wodociągowych stanowią około 30–40% wszystkich robót eksploatacyjnych.

3.2.4 Analiza krytyczności sieci wodociągowej

Mianem awarii określa się uszkodzenie obiektu lub systemu, wpływające w sposób negatywny na jego dalsze funkcjonowanie przez określony czas (np. awaria przewodu wodociągowego, skutkuje brakiem dostawy wody dla określonej grupy odbiorców) [70]. W celu minimalizacji awarii obecne systemy powinny cechować się wysoką skutecznością oraz niezawodnością. W terminologii związanej z analizą ryzyka przez skuteczność rozumie się umiejętność realizacji podstawowych zadań systemu z zachowaniem możliwie jak najniższych kosztów (efektywność) oraz umiejętność realizacji zadań z zachowaniem tzw. odpowiedniości (tj. określonej wydajności, sterowalności oraz kompatybilności) [70]. Niezawodność natomiast oznacza zdolność realizacji założonych funkcji systemu ze spełnieniem wymagań funkcjonalnych i zachowaniem wymagań pod względem bezpieczeństwa [69, 70]. Na rys. 3.4 przedstawiona została propozycja schematu własności systemu.



Rys. 3.4 Schemat pojęciowy własności systemu [70]

Niezawodność systemu cechuje się ogólnie rozumianą gotowością oraz bezpieczeństwem. Z punktu widzenia dyspozycyjności, system zbiorowego zaopatrzenia w wodę powinien być w sposób ciągły zdolny do wykonywania swoich zadań tj. uzdatnienie i dystrybucja wody zdatnej do spożycia. Ponadto system powinien być redundantny pod względem struktury i zasobów oraz obsługiwalny tzn. podatny na modyfikację, rozwój, przywrócenie lub odtworzenie stanu umożliwiającego pracę w określonych warunkach oraz wykrywanie i identyfikację zagrożeń [69]. W przypadku bezpieczeństwa system charakteryzuje się zdolnością unikania zagrożeń poprzez [70]: nieszkodliwość (eliminacja punktów krytycznych poprzez modernizację i zabezpieczenie wrażliwych elementów sieci wodociągowej), ochranialność (przeciwdziałanie szkodliwym sytuacjom wynikającym z otoczenia) oraz niezagrażalność (przysposobienie systemu do niwelowania i ograniczania oddziaływania pobliskiego otoczenia). W systemach zbiorowego zaopatrzenia w wodę obowiązuje elementarna definicja ryzyka, oznaczająca iloczyn prawdopodobieństwa wystąpienia sytuacji niepożądanej i strat powstałych w wyniku jej wystąpienia. Analiza ryzyka to wielostopniowa procedura polegająca na identyfikacji zagrożeń, określaniu prawdopodobieństwa wystąpienia analizowanego przypadku oraz przewidywanie jej skutków [71]. Poziom ryzyka zwiększa się wraz ze wzrostem ww. prawdopodobieństwa. W przypadku systemów zbiorowego zaopatrzenia w wodę, szacowanie ryzyka $r_{security}$ w znaczeniu *security* określone zostało wzorem przedstawionym w równaniu (3.1) [70]:

$$r_{security} = \frac{P \cdot S}{O} \tag{3.1}$$

gdzie:

- P – prawdopodobieństwo wystąpienia awarii (waga punktowa);
- S – straty wynikające z tytułu wystąpienia awarii (waga punktowa);
- O – ochrona systemu przed wystąpieniem awarii (waga punktowa).

Natomiast szacowanie ryzyka r_{safe} w znaczeniu *safe*" określone zostało wzorem przedstawionym w równaniu (3.2) [70]:

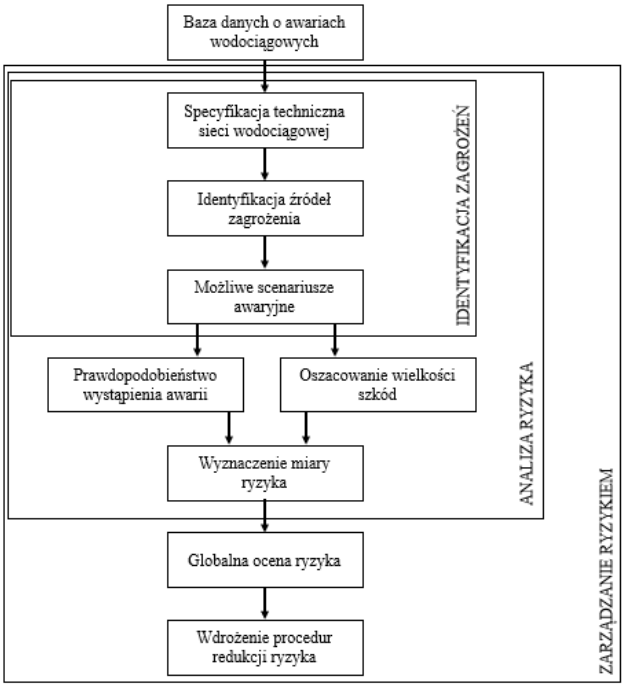
$$r_{safty} = P \cdot S \cdot V \tag{3.2}$$

gdzie:

- P – prawdopodobieństwo wystąpienia awarii (waga punktowa);
- S – straty wynikające z tytułu wystąpienia awarii (waga punktowa);
- V – podatność systemu na wystąpienie awarii (waga punktowa).

W systemach dystrybucji wody podatność związana jest przede wszystkim ze sprawnością usuwania awarii, niezawodnością działania obiektów technicznych, elementów pomiarowych i urządzeń wykonawczych, liczbą stacji uzdatniania wody, liczbą ujęć wody oraz topologią sieci [70]. Bezpieczeństwo tych systemów związane jest natomiast z [70, 92]: monitorowaniem jakości wody, strefami ochrony źródeł wody, zarządzaniem i określaniem parametrów hydraulicznych pracy systemu, dysponowaniem alternatywnymi źródłami zaopatrzenia w wodę zdatną do spożycia (w tym zbiorniki) oraz profesjonalnym zarządzaniem ryzykiem. Na podstawie danych historycznych o awariach występujących w systemie dystrybucji wody oraz informacji o strukturze sieci wodociągowej, następuje identyfikacja możliwych źródeł zagrożenia. Na podstawie ich lokalizacji możliwe jest określenie prawdopodobnych scenariuszy awaryjnych. Analiza ryzyka opiera się więc na prawdopodobieństwie (częstości) wystąpienia zdarzenia niepożądanego oraz oszacowania wielkości szkody powstałej w jej wyniku. W oparciu o te dane wyznacza się tzw. miary ryzyka czyli funkcji przypisującej współczynnik

ryzyka (wyrażane przez liczby rzeczywiste). Zgodnie z jej wartością możliwe staje się zarządzanie ryzykiem poprzez globalną ocenę ryzyka i wdrożenie procedur mających na celu jego redukcję. Na rys. 3.5 przedstawiona została procedura analizy i oceny ryzyka związanego z funkcjonowaniem systemu zbiorowego zaopatrzenia w wodę.



Rys. 3.5 Istota analizy i oceny ryzyka w systemach dystrybucji wody [70]

Podsumowując, analiza ryzyka jest uporządkowanym ciągiem czynności, umożliwiającym identyfikację zagrożenia, planowanie i zarządzanie ryzykiem w celu wyeliminowania go lub obniżenia jego poziomu do wstępnie akceptowalnego. Analiza ryzyka jest fundamentem procesu określania istotności poszczególnych obiektów techniczno-technologicznych w systemie dystrybucji wody. Na podstawie średniej wartości ryzyka eksploatacyjnego (wynikających z cech danego elementu) określa się tzw. krytyczność elementu (np. wpływ awarii konkretnego przewodu na funkcjonowanie całej sieci wodociągowej). Istnieje wiele metod analizy krytyczności opierających się na pojedynczych zdarzeniach (np. metoda FMECA) lub na tzw. rankingach krytyczności. Zarówno w metodach ogólnych (ranking krytyczności) jak i tych bardziej złożonych (FMECA) rezultat analizy krytyczności powinien jednoznacznie wskazać elementy bardziej i mniej krytyczne w analizowanym systemie dystrybucji wody. Analizę

krytyczności poszczególnych elementów wchodzących w system dystrybucji wody przeprowadzić można na podstawie wskaźnika intensywności uszkodzeń λ opisanego równaniem (3.3) [87]:

$$\lambda = \frac{n\,(\Delta t)}{L \cdot \Delta t} \tag{3.3}$$

gdzie:

- $n\,(\Delta t)$ – liczba awarii w okresie (Δt);
- L– długość (w kilometrach) analizowanych przewodów w okresie (Δt);
- (Δt) – rozpatrywany okres (w latach).

W przypadku wykorzystywania modeli matematycznych sieci wodociągowych (których sposób budowy został opisany w Rozdziale 4) do symulacji procesu dystrybucji wody, możliwe jest przeprowadzenie symulacji sytuacji awaryjnej dla wskazanych przewodów. Wyniki symulacji pozwolą na określenie dodatkowego wskaźnika reprezentującego stopień krytyczności przewodu (F_{KP}). Równanie (3.4) przedstawia przykładowe wykorzystanie informacji o stanie sieci w trakcie awarii przewodu do wyznaczenia jego krytyczności:

$$F_{KP} = \frac{D + \,(J_{BW} \cdot 0.5) + (J_{OC} \cdot 0.25) + F_S}{J_{CL}} \tag{3.4}$$

gdzie:

- D – średnica analizowanego przewodu [m];
- J_{BW} – liczba węzłów w sieci wodociągowej pozbawiona dostępu do wody w przypadku wystąpienia awarii na analizowanym odcinku;
- J_{OC} – liczba węzłów, w których ciśnienie wody spadło poniżej zakładanej wartości (węzły pozbawione dostępu do wody nie są wliczane do sumy J_{OC});
- J_{CL} – liczba wszystkich węzłów w analizowanej sieci;
- F_S – spadek przepływu względem wyników wzorcowych.

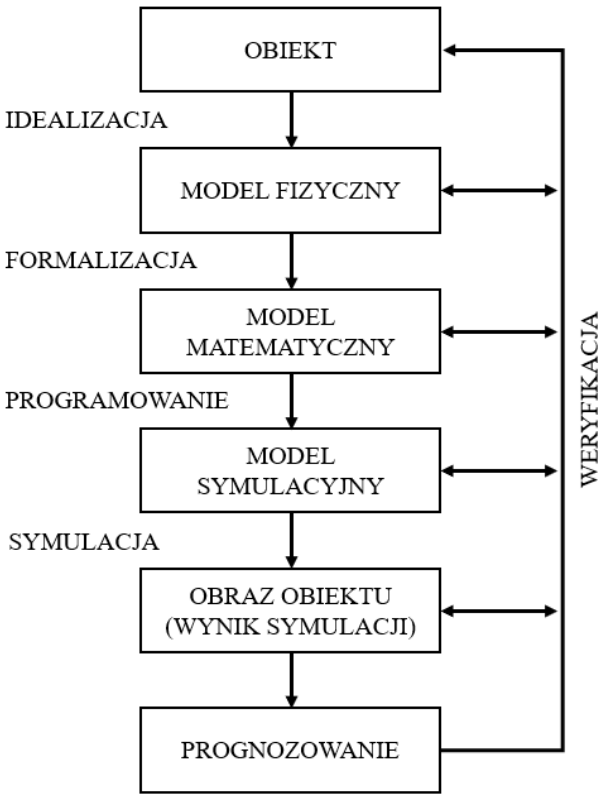
4. Modelowanie procesu dystrybucji wody

4.1 Wprowadzenie

Wielka encyklopedia zarządzania definiuje model jako *układ założeń przyjmowanych w danej nauce w celu realizacji analizowanego problemu badawczego. Jest to hipotetyczna konstrukcja myślowa, będąca uproszczonym obrazem badanego fragmentu rzeczywistości* [116]. Ze względu na ogólne znaczenie tego sformułowania, dokonano klasyfikacji modeli na modele fizyczne i abstrakcyjne, które wyrażone są w odpowiednim języku i posiadają z góry ustaloną strukturę. W przypadku modeli abstrakcyjnych najczęściej używanym modelem jest model matematyczny, który reprezentuje pewien wycinek rzeczywistości, realizowany w określonym celu. W trakcie modelowania wykorzystuje się zbiór operatorów matematycznych oraz symboli, których zadaniem jest jak najlepsze odwzorowanie rzeczywistości [15]. Głównym celem budowy modeli jest możliwość tworzenia symulacji zachowania się obiektu sterowania po wprowadzeniu scenariuszy testowych zawierających różnego rodzaju sygnały sterujące. Jest to niewątpliwie bardzo duża zaleta tego typu rozwiązania. Dzięki wykorzystaniu modeli operator może przygotować się na ewentualne skutki niepożądanych sytuacji lub system tego typu może stanowić wysokiej jakości narzędzie do trenowania przyszłych operatorów i nadzorców obiektów sterowania. Przykłady takich rozwiązań przedstawiono w pracy [85] Nieustanny rozwój techniczny spowodował również wzrost mocy obliczeniowej co umożliwia testowanie złożonych algorytmów sterowania wykorzystujących metody sztucznej inteligencji [14, 110].

Realizacje modelu matematycznego należy rozpocząć od określenia jego celu, a więc definicji problemu (lub grupy problemów) z jakim ma się zmierzyć. Przykładem takiego celu może być wyznaczenie optymalnego punktu pracy urządzeń wykonawczych na sieci wodociągowej lub określenie kolejności naprawy awarii w celu najszybszego przywrócenia zdolności dostawczej sieci wodociągowej [13, 25, 48]. Po określeniu celu, należy zastanowić się nad wyborem wielkości wejściowych i wyjściowych wchodzących w skład całego modelu (w ten sposób określa się stopień autonomiczności modelu względem otoczenia). Kolejnym etapem budowy modelu matematycznego jest agregacja zdobytej wiedzy na temat modelowanego zjawiska, procesu w postaci baz wiedzy. W przypadku niewystarczającej wiedzy na temat funkcjonowania danego zjawiska lub

przebiegu danego procesu wymagane jest przeprowadzenie procedury identyfikacji parametrów (tj. wyznaczanie estymatorów nieznanych parametrów poprzez odpowiednie zmodyfikowanie danych pomiarowych). Sprawdzenie poprawności działania zrealizowanego modelu polega na porównaniu go z dostępnymi wzorcami testowymi (tzw. benchmarking), analizą porównawczą wyników z modelem wzorcowym lub należy przyrównać jego zachowanie do zachowania tego procesu lub jego części w rzeczywistości. Na rys. 4.1 został przedstawiony schemat blokowy procesu modelowania matematycznego.



Rys. 4.1 Schemat blokowy procesu modelowania matematycznego [42]

Jest wiele innych metod opisu struktury sieci wodociągowych. Przykład wykorzystania geometrii fraktalnej do projektowania sieci wodociągowych opisano w pracy [45]. Ze względu na charakter pracy tj. modelowanie sytuacji awaryjnych występujących na sieci wodociągowej po wystąpieniu sytuacji katastroficznej w celu zdobycia informacji o sposobie działania i funkcjonowania sieci, zostaną wykorzystane modele hydrauliczne dostępnych sieci wodociągowych.

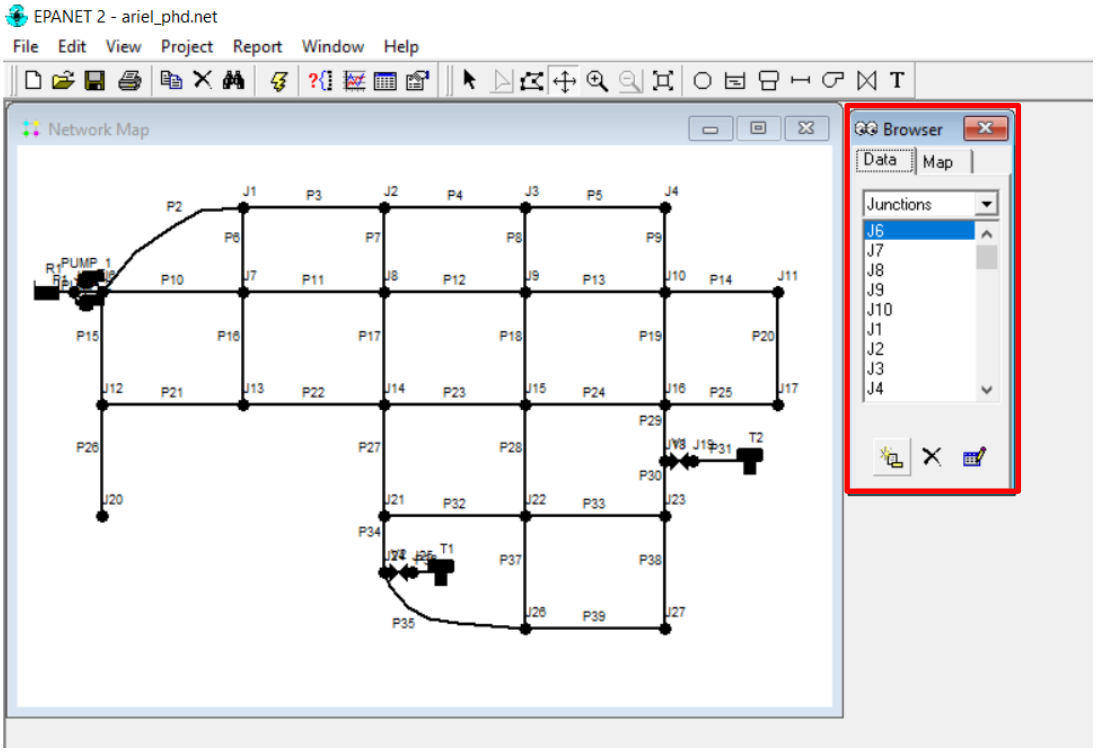
4.2 Narzędzia informatyczne wspomagające modelowanie procesu dystrybucji wody

Pierwszą propozycją wykorzystania modelu matematycznego do obliczeń przepływów w przewodach sieci wodociągowej była praca autorstwa Hardiego Crossa opublikowana w 1963 roku w University of Illinois at Urbana-Champing pt. „*Analysis of Flow in Networks of Conduits or Conductors*". Po około czterdziestu latach, w dobie rozwoju komputerów, badań operacyjnych i modelowania komputerowego zostały zaproponowane dwa podejścia autorstwa: R. M. Clark, R. M. Males W. M. Grayma oraz J.A. Coyle. Pierwsza propozycja dotyczyła modelowania procesu dystrybucji wody w stanie ustalonym (artykuł pt. „*Predicting Water Quality in Distribution Systems*"), natomiast druga, która była następstwem pierwszej umożliwiała modelowanie zmian jakości wody w czasie (artykuł pt. „*Modelling Distribution System Water Quality Dynamic Aproach*")[15].

4.2.1 EPANET

Prace naukowe wymienione we wprowadzeniu tego podrozdziału przyczyniły się do powstania programu komputerowego o nazwie EPANET. Dzięki wykorzystaniu mocy obliczeniowej komputerów możliwe stało się wykonanie symulacji działania systemu zbiorowego zaopatrzenia w wodę w warunkach ustalonych oraz w warunkach dynamicznych. W przypadku trybu ustalonego wszystkie parametry analizowanej sieci wodociągowej są stałe w czasie (w tym wartości zapotrzebowania na wodę). Umożliwia to sprawdzenie stanu sieci w określonym momencie. W przypadku drugim parametry analizowanej sieci zmieniają się w sposób dynamiczny (np. poziom wody w zbiornikach). Taka funkcjonalność pozwala na symulację następujących po sobie warunków ustalonych, w których każdy stan zawiera odrębny zbiór rozwiązań. Symulacja polega na rozwiązywaniu złożonych układów równań, w skład których wchodzą równania zachowania masy (równania liniowe) oraz równania zachowania energii (równania nieliniowe). W tym celu wykorzystuje się metodę pierścieniowo-węzłową w połączeniu z metodą gradientową [77]. Uważa się, że wykorzystanie potencjału komputerowego modelowania systemów zbiorowego zaopatrzenia w wodę powinno być integralną częścią procesu zarządzania, planowania i modernizacji sieci wodociągowych [97]. Aktualnie oprogramowanie to rozwijane jest przez Agencję Ochrony Środowiska USA

(ang. *U. S. Environmental Protection Agency*) i jest dostępne na zasadach licencji publicznej (ang. *Public Domain*), umożliwiającej wykorzystywanie zarówno samego programu komputerowego jak i jego kodów źródłowych. Interfejs programu EPANET został przedstawiony na rys. 4.2.



Rys. 4.2 Interfejs programu EPANET [opracowanie własne]

Modele hydrauliczne sieci wodociągowych reprezentowane są za pośrednictwem węzłów (ang. *nodes*), które mogą symbolizować: punkty poboru wody (ang. *junction*), zbiorniki (ang. *tanks*), rezerwuary (ang. *reservouirs*), źródła (ang. *sources*), oraz przewodów (ang. *links*) reprezentujących takie elementy jak rury (ang. *pipes*), zasuwy (ang. *valves*) oraz pompy (ang. *pumps*). W sytuacji konieczności utworzenia modelu hydraulicznego istniejącego systemu dystrybucji wody należy posiadać informację umożliwiające wierne odwzorowanie topologii całej sieci. Informacje te dotyczą: rozmieszczenia punktów poboru wody, urządzeń wykonawczych, zbiorników oraz informacji na temat wzorców rozbioru w poszczególnych punktach czy rzędnych terenu rozmieszczonych obiektów wodociągowych. Jest to bardzo istotne, ponieważ jak już

wspomniano obliczenia przepływu wody w przewodach wodociągowych oparte są na prawie zachowania masy i energii [21].

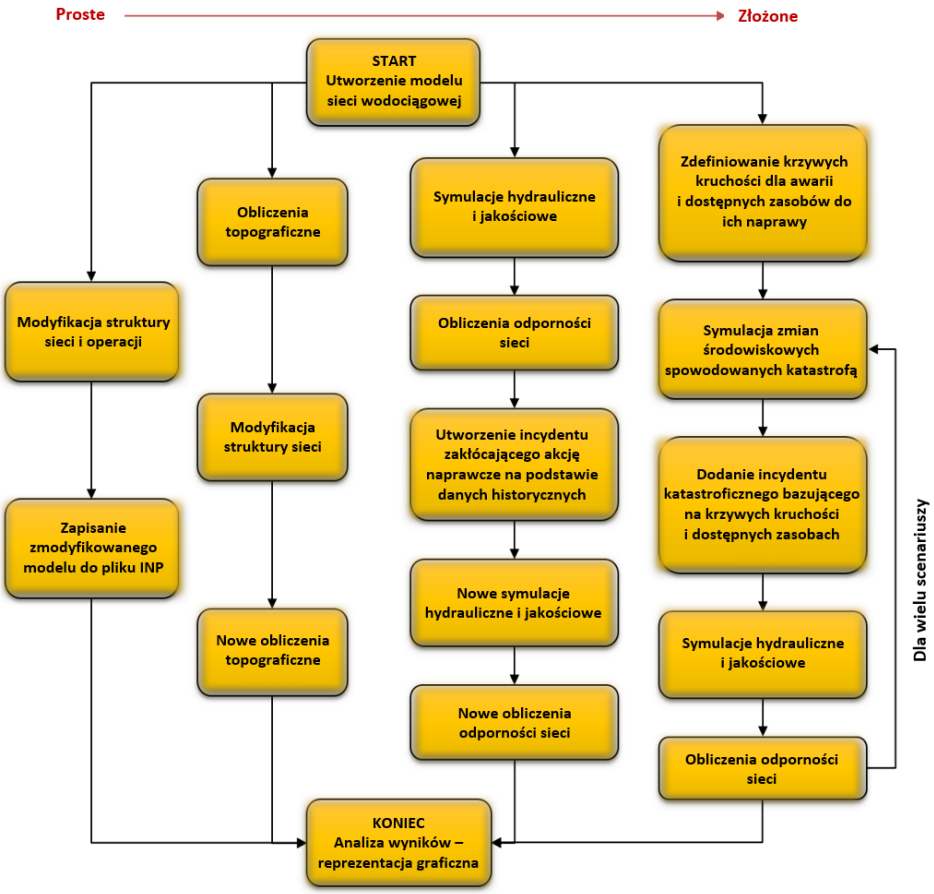
4.2.2 The Water Network Tool for Resilience (WNTR)

Systemy zbiorowego zaopatrzenia w wodę zmagają się z wieloma problemami, których zbagatelizowanie może negatywnie wpłynąć na ich dalsze funkcjonowanie. Przykładem takich sytuacji może być: pogorszenie jakości wody, starzejąca się infrastruktura powodująca częstsze powstawanie awarii, klęska żywiołowa, sytuacje katastroficzne, cyberataki, a nawet terroryzm [38, 39, 40]. Ważna jest więc zdolność do przewidywania zachowania systemu dystrybucji wody w sytuacji wystąpienia pewnej liczby zakłóceń w celu ich eliminacji. Narzędzia do symulacji i analizy mogą pomóc przedsiębiorstwom wodociągowym sprawdzić zdolności dostawcze ich sieci oraz ich krytyczność w pewnych określonych sytuacjach.

Przykładem takiego narzędzia jest biblioteka języka Python o nazwie WNTR (akronim nazwy *The Water Network Tool for Resilience*). Biblioteka ta została zaprojektowana w celu symulacji pracy różnorodnych sieci wodociągowych oraz analizy ich odporności na różne czynniki zewnętrzne. Za pośrednictwem dostępnego API, WNTR pozwala na zmianę struktury sieci w łatwy sposób. Dodatkowo taka funkcjonalność umożliwia tworzenie w sposób automatyczny, scenariuszy testujących zachowanie sieci przy określonych warunkach początkowych (wraz z symulacją incydentów zakłócających poprawne działanie sieci) [6, 38, 39, 40]. WNTR wykorzystuje silnik obliczeniowy narzędzia EPANET. Oznacza to, że umożliwia wiele tożsamych funkcji co EPANET (przede wszystkim umożliwia symulację hydrauliczną oraz jakościową).

Metoda obliczeniowa EPANETu (tzw. EPANETSimulator) pozwala jedynie na wykonywanie symulacji uzależnionych od zapotrzebowania tzw. DD (ang. *Demand Driven*), dlatego WNTR zaproponował własny silnik obliczeniowy (tzw. WNTRSimulator) [4, 6, 38, 39]. Zaproponowane podejście pozwala na przeprowadzanie symulacji uzależnionych od ciśnienia tzw. PDD (ang. *Pressure Demand Driven*) oraz modelowanie sytuacji takich jak braki w dostawie zasilania, trzęsienia ziemi, wycieki, przypadkowe zdarzenia skrzenia wody, itp. [39, 40]. Na podstawie scenariusza, WNTR

generuje model sieci, przeprowadza symulację probabilistyczną i zwraca wyniki do analizy (z możliwością generowania grafów przepływu i innych niezbędnych w analizie grafik). Na rys. 4.3 przedstawiono przykładowe przypadki użycia narzędzia WNTR. Jak widać biblioteka pozwala na wykonywanie prostych operacji typu modyfikacji topologii systemu wodociągowego i zapisu go do pliku .INP[1], oraz złożonych zagnieżdżonych symulacji hydraulicznych oraz jakościowych.



Rys. 4.3 Graf przykładowych przypadków użycia WNTR [117]

4.3 Modelowanie sieci wodociągowych

Zgodnie z zasadami tworzenia modeli matematycznych, przed rozpoczęciem budowy takiego modelu należy jasno określić jego cel. Z punktu widzenia operatorów sieci wodociągowych i całego przedsiębiorstwa WOD-KAN najczęstszym powodem realizacji modeli hydraulicznych sieci wodociągowych jest możliwość projektowania

[1] Plik o rozszerzeniu INP jest plikiem typu tekstowego o określonej strukturze przechowujący wszystkie niezbędne informacje potrzebne do zamodelowania procesu dystrybucji

nowych odcinków sieci, symulacja działania sieci po próbie jej modernizacji, sprawdzenie warunku ciśnień w przypadku dodania nowego wzorca zapotrzebowania czy diagnostyka (np. analiza krytyczności sieci wodociągowej). Ponadto osoba nadzorująca proces może przeprowadzić symulację pracy sieci wodociągowej w celu realizacji celu ekonomicznego, ekologicznego czy organizacyjnego (szerzej omówione w rozdziale poświęconemu warstwowej strukturze sterownia – patrz Rozdział 2.2). Realizacja złożonego modelu hydraulicznego wymaga od projektanta wprowadzenia informacji takich jak charakterystyka przewodów, pomp, zbiorników, zasuw, parametrów symulacji, informacji topograficznych oraz danych na temat wzorców rozbioru w poszczególnych węzłach sieci.

4.3.1 Charakterystyka przykładowego modelu sieci wodociągowej

W celu objaśnienia sposobu modelowania sieci wodociągowej, podrozdział ten został poświęcony przedstawieniu przykładowej charakterystyki (nie rzeczywistego) systemu dystrybucji wody opracowanego przez autora niniejszej rozprawy. W skład odwzorowywanego modelu testowego wchodzą takie elementy jak: rezerwuar (reprezentujący stację uzdatniania wody), dwa zbiorniki wyrównawcze, dwadzieścia siedem węzłów, trzydzieści dziewięć przewodów, dwie pompy oraz dwie zasuwy. W przypadku sieci wodociągowych modelowanie rozpoczyna się od elementów przedstawionych jako węzły tj. węzły poboru wody (ang. *Junction*), stacje uzdatniania wody, reprezentowane jako rezerwuary (ang. *Reservouir*) oraz zbiorniki (ang. *Tanks*). Aby zamodelować rezerwuar (tzw. zbiornik źródłowy) potrzeba następujących informacji:

- identyfikator elementu (ang. *Label*);
- lokalizacja (współrzędne, ang. *Coordinates*);
- rzędna zwierciadła wody w zbiorniku (ang. *Total Head*).

Opcjonalnie można dodać również wzorzec wysokości (ang. *Head Pattern*), który zawiera zbiór przeliczników wartości rzędnej zwierciadła w zbiorniku. W analizowanym przykładzie zamodelowany rezerwuar będzie miał identyfikator *R1*. Zlokalizowany będzie w miejscu o współrzędnych (0.0, 80.0), natomiast rzędna zwierciadła wody

w zbiorniku będzie wynosić 150 [ft]. W przypadku zbiorników wyrównawczych (ang. *Tanks*) dane niezbędne do zamodelowania ich pracy to [60]:

- identyfikator elementu (ang. *Label*);
- współrzędne (ang. *Coordinates*);
- rzędna dna zbiornika (ang. *Elevation*);
- poziom początkowy (ang. *Initial Level*);
- poziom minimalny (ang. *Minimum Level*);
- poziom maksymalny (ang. *Maximum Level*);
- średnica zbiornika (ang. *Diameter*).

Poziom początkowy, minimalny i maksymalny określa kolejno: początkową, minimalną oraz maksymalną wysokość zwierciadła wody w stosunku do dna zbiornika. Informacje potrzebne do zamodelowania zbiorników przedstawione zostały w Tabeli 4.1.

Tab. 4.1 *Informacje o zbiornikach w modelowanym systemie [opracowanie własne]*

Informacje o zbiornikach (typ: Tank)							
Identyfikator [ID]	**Współrzędne**		**Rzędna terenu**	**Poziom początkowy** [ft]	**Poziom minimalny** [ft]	**Poziom maksymalny** [ft]	**Średnica** [ft]
	x	**y**					
T1	70.0	30.0	132.0	13.5	1.0	32.5	85.0
T2	125.0	50.0	129.0	29.0	5.0	40.0	165.0

Węzły w modelowanym systemie wodociągowym reprezentują odbiorców, do których przypisane zostają rzeczywiste wzorce rozbiorów wody. W celu odwzorowania powyższych elementów w przykładowym modelu sieci dystrybucji wody wymagana jest wiedza na temat:

- identyfikatora elementu (ang. *Label*);
- współrzędnych (ang. *Coordinates*);
- rozbioru podstawowego (ang. *Base Demand*);
- wzorcu rozbioru (ang. *Demand Pattern*);
- (opcjonalnie) kategorii rozbiorów (ang. *Demand Categories*).

Na podstawie wzorca rozbioru obliczana jest średnia wartość rozbioru w poszczególnych godzinach dnia. Kategoria rozbioru zawiera identyfikator grupy odbiorców. Oprogramowanie EPANET umożliwia określenie podziału grupy odbiorców np. na grupę odbiorców indywidualnych, zakłady przemysłowe, infrastrukturę krytyczną itp. Informacje potrzebne do zamodelowania węzłów przedstawione zostały w Tabeli 4.2.

Tab. 4.2 *Informacje o przykładowych węzłach w modelowanym systemie [opracowanie własne]*

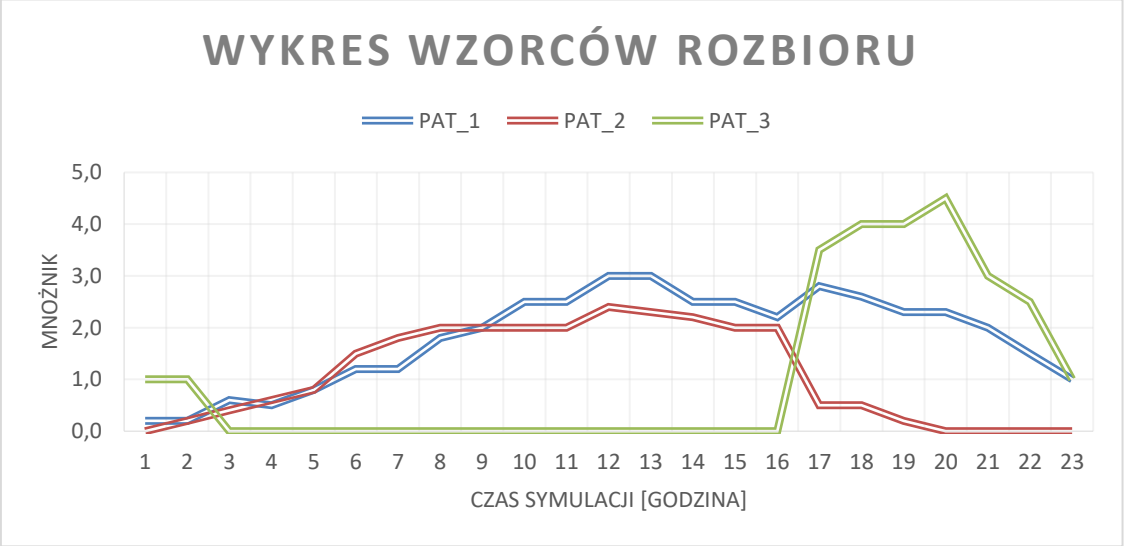
Informacje o węzłach (typ: Junction)						
Identyfikator [ID]	**Współrzędne**		**Rzędna** [ft]	**Rozbiór podstawowy** [GPM]	**Wzorzec rozbioru** [ID]	**Kategoria rozbioru** [ID]
	x	**y**				
J1	35.0	95.0	20.0	30.0	PAT_1	CAT_1
J2	60.0	95.0	22.0	30.0	PAT_1	CAT_1
J3	85.0	95.0	24.0	33.0	PAT_1	CAT_1
J4	110.0	95.0	30.0	24.0	PAT_1	CAT_1
J5	5.0	80.0	25.0	0.0	PAT_1	CAT_3
J6	10.0	80.0	25.0	0.0	PAT_1	CAT_3
J7	35.0	80.0	23.0	55.0	PAT_1	CAT_1
J8	60.0	80.0	23.0	55.0	-	CAT_1
J9	85.0	80.0	27.5	40.0	PAT_1	CAT_1
J10	110.0	80.0	31.0	42.0	PAT_1	CAT_1
J11	130.0	80.0	35.0	25.0	PAT_2	CAT_2
J12	10.0	60.0	22.0	48.0	-	CAT_1
J13	35.0	60.0	28.0	50.0	PAT_1	CAT_1
J14	60.0	60.0	30.0	47.0	PAT_1	CAT_1
J15	85.0	60.0	30.0	42.0	PAT_1	CAT_1
J16	110.0	60.0	33.0	20.0	PAT_1	CAT_1
J17	130.0	60.0	35.0	15.0	PAT_1	CAT_1
J18	111.0	50.0	40.0	0.0	PAT_1	CAT_3
J19	115.0	50.0	40.0	0.0	PAT_1	CAT_3
J20	10.0	40.0	45.0	80.0	PAT_3	CAT_4
J21	60.0	40.0	35.0	35.0	-	CAT_1
J22	85.0	40.0	37.0	40.0	-	CAT_1
J23	111.0	40.0	40.0	21.0	PAT_2	CAT_2
J24	60.0	30.0	37.0	0	PAT_1	CAT_3
J25	65.0	30.0	37.0	0	PAT_1	CAT_3
J26	85.0	20.0	35.0	30	PAT_1	CAT_1
J27	11.0	20.0	36.5	40.0	PAT_1	CAT_1

Na podstawie analizy danych zawartych w powyższej tabeli, zauważyć można, że w modelowanym systemie dystrybucji wody zaproponowano trzy wzorce rozbioru o identyfikatorach *PAT_1, PAT_2* oraz *PAT_3* oraz cztery kategorie rozbioru: *CAT_1*,

CAT_2, *CAT_3* oraz *CAT_4*. Kategorie rozbiorów ułatwiają rozpoznanie poszczególnych odbiorców w systemie zbiorowego zaopatrzenia w wodę. W analizowanym przykładzie kategorie oznaczają:

- *CAT_1* – mieszkańców oraz drobne zakłady przemysłowe;
- *CAT_2* – średnie zakłady przemysłowe;
- *CAT_3* – węzły przyłączeniowe (brak odbiorców);
- *CAT_4* – konkretny zakład przemysłowy o nazwie *XYZ*.

W przypadku wzorców, pierwszy (*PAT_1*) odzwierciedla zużycie wody mieszkańców oraz niewielkich przedsiębiorstw. *PAT_2* odpowiada natomiast wzorcowi poboru wody przez średnie zakłady przemysłowe, które mają podpisaną umowę z przedsiębiorstwem wodno-kanalizacyjnym na ciągłą dostawę wody w określonych porach dnia. Ostatni wzorzec dotyczy większego przedsiębiorstwa, którego proces produkcyjny odbywa się w godzinach wieczornych. W przypadku braku informacji o wzorcu rozbioru dla poszczególnych węzłów, do obliczeń wykorzystywany zostaje wzorzec domyślny określony w ustawieniach symulacji sieci wodociągowej. Wykres rozbioru poszczególnych wzorców rozbioru został przedstawiony na rys. 4.4.



Rys. 4.4 Wykres rozbioru w modelowanym systemie dystrybucji [opracowanie własne]

W Tabeli 4.3 zawarto informacje potrzebne do zamodelowania wzorców rozbioru w analizowanym przykładzie sieci. Na podstawie zapotrzebowania bazowego i wzorca rozbioru określane jest zużycie wody w określonej godzinie symulacji.

Tab. 4.3 *Informacje o wzorcach rozbioru w modelowanym systemie [opracowanie własne]*

Informacje o wzorcach rozbioru (typ: Demand Pattern)	
Identyfikator [ID]	**Mnożniki** [lista przeliczników rozbiorów dla danych godzin]
PAT_1	[0.2, 0.2, 0.6, 0.5, 0.8, 1.2, 1.2, 1.8, 2.0, 2.5, 2.5, 3.0, 3.0, 2.5, 2.5, 2.2, 2.8, 2.6, 2.3, 2.3, 2.0, 1.5, 1.0]
PAT_2	[0.0, 0.2, 0.4, 0.6, 0.8, 1.5, 1.8, 2.0, 2.0, 2.0, 2.0, 2.4, 2.3, 2.2, 2.0, 2.0, 0.5, 0.5, 0.2, 0.0, 0.0, 0.0, 0.0]
PAT_3	[1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.5, 4.0, 4.0, 4.5, 3.0, 2.5, 1.0]

Jak wspomniano we wstępie rozdziału, kolejność modelowania poszczególnych elementów sieci wodociągowej ma znaczenie, w szczególności gdy mówimy o węzłach i przewodach. Jest to istotne, ponieważ w sytuacji odwzorowywania elementów przedstawionych jako przewody (ang. *Links*) należy podać identyfikatory węzłów, do których są one podłączone. W środowisku programistycznym EPANET w formie przewodów przedstawiane są elementy takie jak: rurociąg (ang. *Pipe*), zasuwa (ang. *Valve*) czy pompa (ang. *Pump*). W przypadku modelowania odcinków (rurociągów) potrzeba następujących informacji:

- identyfikator elementu (ang. *Label*);
- identyfikator węzła początkowego (ang. *Start Node*);
- identyfikator węzła końcowego (ang. *End Node*);
- długość odcinka (ang. *Length*);
- średnica przewodu (ang. *Diameter*);
- chropowatość przewodu (ang. *Roughness*);
- współczynnik oporów miejscowych (ang. *Loss Coefficient*);
- status początkowy (ang. *Initial Status*).

Tab. 4.4 *Informacje o przewodach w modelowanym systemie [opracowanie własne]*

Informacje o przewodach (typ: Pipe)							
Identyfikator [ID]	**Węzeł początkowy** [ID]	**Węzeł końcowy** [ID]	**Długość** [ft]	**Średnica** [in]	**Chropowatość**	**Straty miejscowe**	**Status** [0/1]
P1	R1	J5	250	24	100	0	1
P2	J1	J6	1200	12	100	0	1
P3	J2	J1	1000	12	100	0	1
P4	J3	J2	1000	12	100	0	1
P5	J4	J3	1000	12	100	0	1
P6	J1	J7	650	10	100	0	1
P7	J2	J8	650	10	100	0	1
P8	J3	J9	650	10	100	0	1
P9	J4	J10	650	10	100	0	1
P10	J6	J7	1000	24	100	0	1
P11	J7	J8	1000	24	100	0	1
P12	J8	J9	1000	24	100	0	1
P13	J9	J10	1000	12	100	0	1
P14	J10	J11	1000	12	100	0	1
P15	J12	J6	1000	18	100	0	1
P16	J13	J7	1000	12	100	0	1
P17	J14	J8	1000	12	100	0	1
P18	J9	J15	1000	24	100	0	1
P19	J6	J10	1000	12	100	0	1
P20	J11	J17	1000	12	100	0	1
P21	J12	J13	1000	12	100	0	1
P22	J13	J14	1000	12	100	0	1
P23	J14	J15	1000	12	100	0	1
P24	J16	J15	1000	18	100	0	1
P25	J17	J16	1000	12	100	0	1
P26	J12	J20	1000	18	100	0	1
P27	J21	J14	1000	12	100	0	1
P28	J15	J22	1000	24	100	0	1
P29	J16	J18	500	18	100	0	1
P30	J18	J23	500	12	100	0	1
P31	T2	J19	600	18	100	0	1
P32	J22	J21	1000	18	100	0	1
P33	J22	J23	1000	18	100	0	1
P34	J21	J24	500	12	100	0	1
P35	J24	J26	1100	12	100	0	1
P36	J25	T1	350	12	100	0	1
P37	J26	J22	1000	18	100	0	1
P38	J23	J27	1000	12	100	0	1
P39	J27	J26	1000	12	100	0	1
P40	J5	J6	250	24	100	0	1

Informacje potrzebne do zamodelowania przewodów przedstawione zostały w Tabeli 4.4. W przypadku definicji pompy, poza informacjami potrzebnymi do zamodelowania przewodu potrzeba również informacji o charakterystyce przepływu (ang. *Pump Curve*) opisującej krzywą wydajności pompy oraz (opcjonalnie):

- moc napędu wyrażoną w kilowatach lub koniach mech. (ang. *Power*);
- wzorzec prędkości obrotowej (ang. *Pattern*) lub względną prędkość obrotową silnika (ang. *Speed*);
- charakterystykę sprawności pompy (ang. *Efficiency Curve*);
- jednostkowy koszt energii elektrycznej (ang. *Energy Price*);
- wzorzec zmienności ceny energii (ang. *Price Pattern*).

W Tabeli 4.5 zawarto informacje potrzebne do zamodelowania pomp w analizowanym przykładzie sieci wodociągowej.

Tab. 4.5 *Informacje o pompach w modelowanym systemie [opracowanie własne]*

Informacje o pompach (typ: Pump)			
Identyfikator [ID]	**Węzeł początkowy** [ID]	**Węzeł końcowy** [ID]	**Parametry**
PUMP_1	J5	J6	HEAD 1
PUMP_2	J5	J6	HEAD 2

Charakterystykę przepływu określa się wykorzystując metodę jednego, trzech lub wielu punktów. W przypadku metody jedno-punktowej pozostałe punkty wyznaczane są w sposób następujący: natężenie przepływu (wartość równa 0), wysokość podnoszenia zwiększona o 133 procent (na podstawie wartości podanej) oraz natężenie przepływu (dwukrotność podanej wartości). Na podstawie tych punktów krzywa obliczana jest na podstawie równania kwadratowego. Metoda trzy-punktowa na podstawie rozmieszczenia punktów takich jak: natężenie przepływu (wartość równa 0), punkt pracy pompy oraz punkt maksymalnego natężenia przepływu przybliża (próbkuje) wartości krzywej w oparciu o równanie kwadratowe. Metoda wielopunktowa wyznacza krzywą na podstawie dowolnej liczby (większej od trzech) punktów. Tabela 4.6 zawiera informacje

potrzebne do zamodelowania krzywych wykorzystywanych przez pompy w analizowanym przykładowym systemie zaopatrzenia.

Tab. 4.6 *Informacje o krzywych w modelowanym systemie [opracowanie własne]*

Informacje o krzywych (typ: Curve)	
Identyfikator [ID]	**Współrzędne punktów** [lista współrzędnych]
1	[(0, 104), (2000, 92), (4000, 63)]
2	[(0, 100), (8000, 60), (14000, 30)]

Ostatnim wykorzystywanym elementem w modelowaniu sieci wodociągowych w środowisku EPANET są zasuwy. Istnieje możliwość odwzorowania pracy sześciu różnych typów zasuw, w tym [60]:

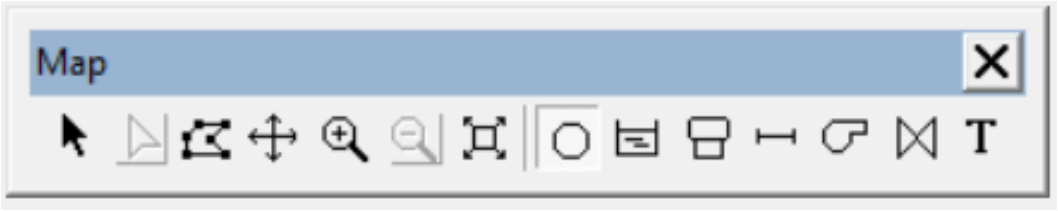
- stabilizujących ciśnienie PSV (ang. *Pressure Stabilizing Valve*);
- redukujących ciśnienie PRV (ang. *Pressure Reducing Valve*);
- zmieniających ciśnienie PBV (ang. *Pressure Breaker Valve*);
- redukujących przepływ FCV (ang. *Flow Control Valve*);
- dławiących TCV (ang. *Throttle Control Valve*);
- ogólnego przeznaczenia GPV (ang. *General Purpose Valve*).

W zależności od przeznaczenia zasuwy, poza informacją o średnicy przewodu, identyfikatorach węzła początkowego oraz końcowego, należy podać informację o statusie zaworu (ang. *Status*) lub jego ustawieniach (ang. *Setting*). Zastosowanie poszczególnych zasuw w procesie dystrybucji wody zostało szerzej opisane w pracy [77]. W modelowanym systemie zbiorowego zaopatrzenia w wodę wykorzystano dwie zasuwy typu PSV o identyfikatorach *V1* oraz *V2*. Pierwsza zasuwa wykorzystuje przewód o średnicy osiemnastu cali, natomiast druga dwunastu.

4.3.2 Modelowanie sieci wodociągowej z wykorzystaniem środowiska EPANET

W celu rozpoczęcia tworzenia modelu hydraulicznego sieci wodociągowej w środowisku EPANET należy utworzyć nowy projekt. W momencie jego utworzenia

ukaże się tzw. pole robocze, na które można zacząć nanosić elementy sieci wodociągowej. Przed rozpoczęciem nanoszenia elementów na obszar roboczy zaleca się ustawienie jego wymiarów (za pomocą *View/Dimensions*). W analizowanym przykładzie, który nie odzwierciedla żadnego rzeczywistego modelu (brak konieczności dopasowania modelu do dostępnych modeli odniesienia np. WGS-84) ustawiono pole robocze o wymiarach 100 na 100 jednostek. Poszczególne elementy przenosi się na obszar roboczy wykorzystując pasek narzędzi, który został przedstawiony na rys. 4.5.



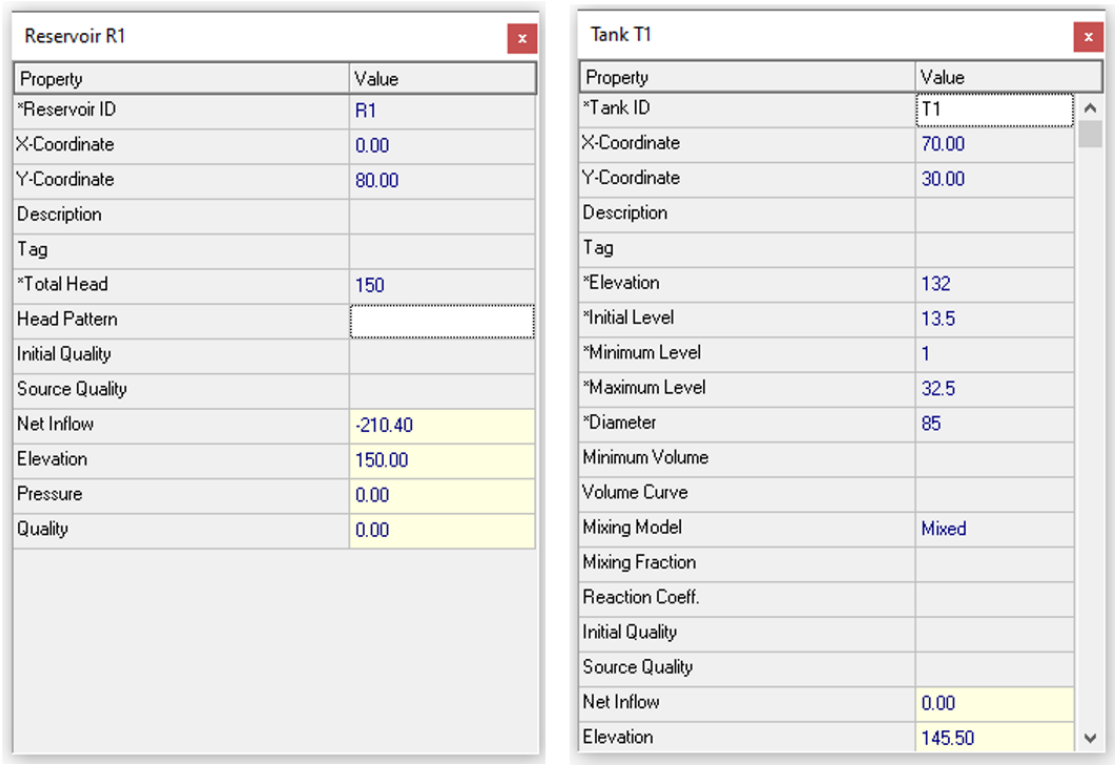
Rys. 4.5 Pasek narzędzi programu EPANET [opracowanie własne]

Pasek narzędzi programu EPANET umożliwia (od lewej):

- zaznaczenie obiektu (ang. *Select Object*);
- zaznaczenie wierzchołka (ang. *Select Vertex*);
- zaznaczenie regionu (ang. *Select Region*);
- przesunięcie (ang. *Pan*);
- powiększenie (ang. *Zoom In*);
- pomniejszenie (ang. *Zoom Out*);
- dopasowanie (ang. *Full Extent*);
- dodanie węzła poboru (ang. *Add Junction*);
- dodanie rezerwuaru (ang. *Add Reservoir*);
- dodanie zbiornika (ang. *Add Tank*);
- dodanie przewodu (ang. *Add Pipe*);
- dodanie pompy (ang. *Add Pump*);

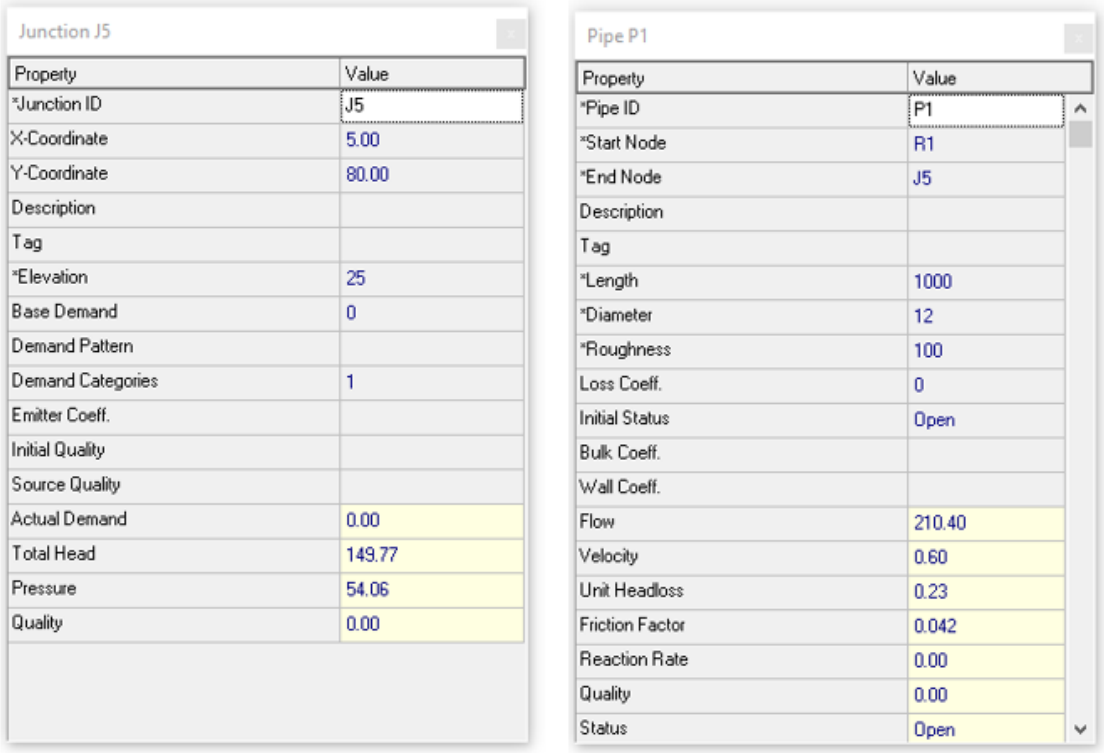
- dodanie zasuwy (ang. *Add Valve*);
- dodanie etykiety (ang. *Add Label*).

Po wybraniu odpowiedniej opcji (np. dodaj rezerwuar lub dodaj zbiornik) i umieszczeniu go w polu roboczym pojawia się ekran umożliwiający wprowadzenie danych o modelowanym elemencie. Rys. 4.6 przedstawia modelowanie rezerwuaru o identyfikatorze *R1* oraz zbiornika o identyfikatorze *T1*.



Rys. 4.6 Modelowanie rezerwuaru i zbiornika w programie EPANET [opracowanie własne]

Opcja „dodaj węzeł” umożliwia umieszczanie węzłów na obszarze roboczym. W sytuacji gdy węzły zostały zamodelowane można odwzorować elementy wchodzące w skład sieci wodociągowej, które są reprezentowane jako odcinki. Rys. 4.7 zawiera przykład odzwierciedlenia węzła o identyfikatorze *J5* oraz przewodu *P1*.



Junction J5

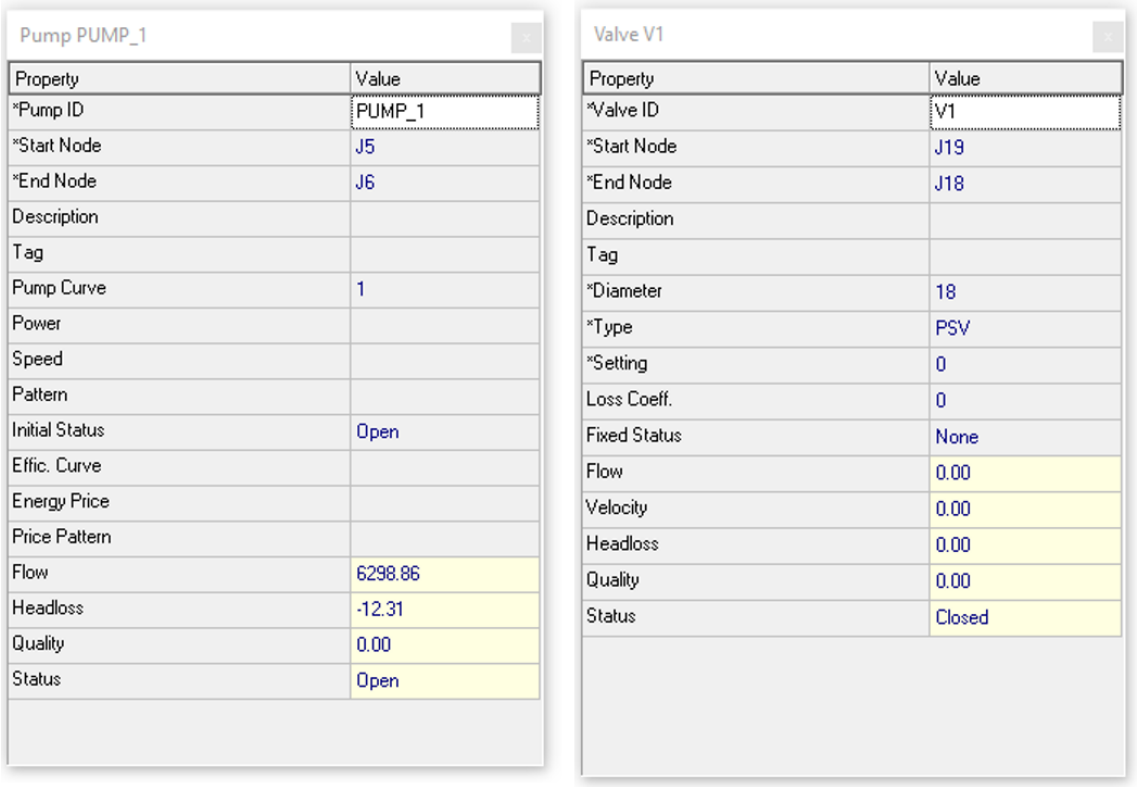
Property	Value
*Junction ID	J5
X-Coordinate	5.00
Y-Coordinate	80.00
Description	
Tag	
*Elevation	25
Base Demand	0
Demand Pattern	
Demand Categories	1
Emitter Coeff.	
Initial Quality	
Source Quality	
Actual Demand	0.00
Total Head	149.77
Pressure	54.06
Quality	0.00

Pipe P1

Property	Value
*Pipe ID	P1
*Start Node	R1
*End Node	J5
Description	
Tag	
*Length	1000
*Diameter	12
*Roughness	100
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	210.40
Velocity	0.60
Unit Headloss	0.23
Friction Factor	0.042
Reaction Rate	0.00
Quality	0.00
Status	Open

Rys. 4.7 Modelowanie węzła poboru oraz przewodu w programie EPANET [opracowanie własne]

Bardziej doświadczone osoby, które zajmują się modelowaniem od lat, odwzorowują sieci fragmentarycznie. Oznacza to, że nie modelują od razu wszystkich węzłów, które wchodzą w skład sieci, tylko pewną część. Później umieszczają przewody lub inne elementy w taki sposób, aby otrzymać fragment działającej sieci. Następnie sukcesywnie powiększają sieć o kolejne węzły i przewody wodociągowe. Ostatnim krokiem tworzenia topologii sieci wodociągowej jest odwzorowanie takich elementów wykonawczych jak pompa czy zasuwa. Na rys. 4.8 przedstawiono przykład modelowania ww. elementów.



Pump PUMP_1

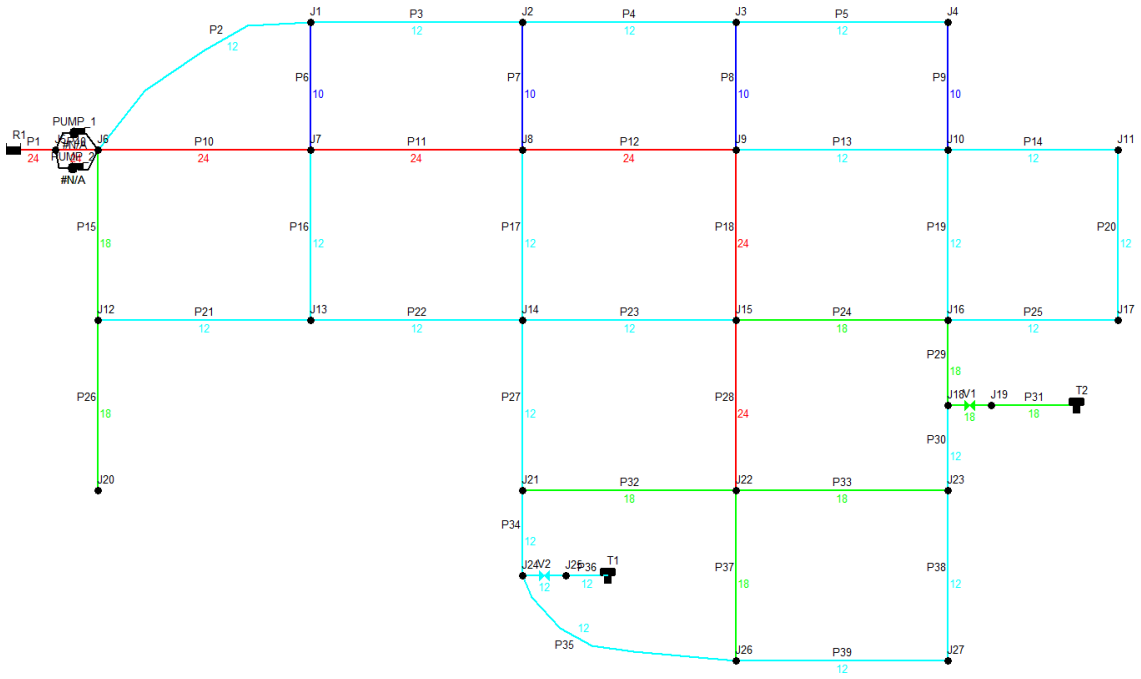
Property	Value
*Pump ID	PUMP_1
*Start Node	J5
*End Node	J6
Description	
Tag	
Pump Curve	1
Power	
Speed	
Pattern	
Initial Status	Open
Effic. Curve	
Energy Price	
Price Pattern	
Flow	6298.86
Headloss	-12.31
Quality	0.00
Status	Open

Valve V1

Property	Value
*Valve ID	V1
*Start Node	J19
*End Node	J18
Description	
Tag	
*Diameter	18
*Type	PSV
*Setting	0
Loss Coeff.	0
Fixed Status	None
Flow	0.00
Velocity	0.00
Headloss	0.00
Quality	0.00
Status	Closed

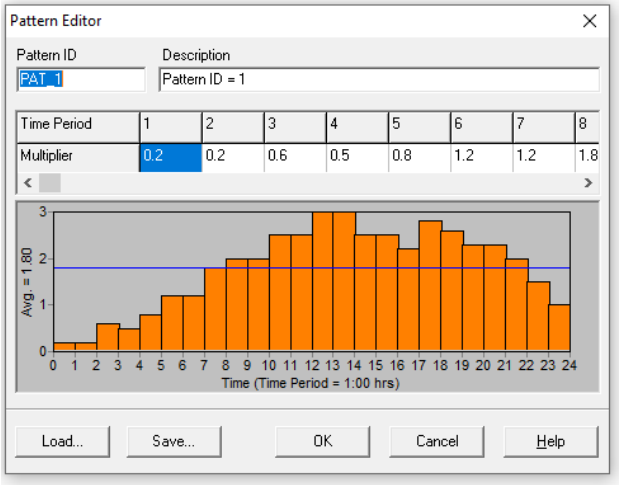
Rys. 4.8 Modelowanie pompy oraz zasuwy w programie EPANET [opracowanie własne]

W celu zamodelowania całej sieci wodociągowej należy wykonać powyższe czynności dla wszystkich elementów wchodzących w skład modelowanej sieci. Przykładowy system dystrybucji wody należy do grupy tzw. mniejszych sieci wodociągowych (sumarycznie zawiera siedemdziesiąt trzy elementy, w skład których wchodzą: rezerwuar, dwa zbiorniki, dwadzieścia siedem węzłów, trzydzieści dziewięć przewodów, dwie zasuwy oraz dwie pompy). Mimo niewielkiego rozmiaru sieci, proces modelowania jest czasochłonny i żmudny. Należy na bieżąco analizować i sprawdzać poprawność danych, ponieważ przy takiej liczbie ręcznie wprowadzanych danych istnieje wysokie prawdopodobieństwo pomyłki. Rezultat modelowania przykładowej sieci wodociągowej został przedstawiony na rys. 4.9. Topologia prezentowanej sieci wodociągowej ma charakter obwodowy. Kolorem czerwonym zaznaczono przewody główne (magistralne), których średnica wynosi 24 cale. Kolorem zielonym oznaczono przewody rozdzielcze (średnica 18 cali), natomiast kolorem jasno niebieskim oraz niebieskim przewody o mniejszej średnicy tj. 12 i 10 cali.



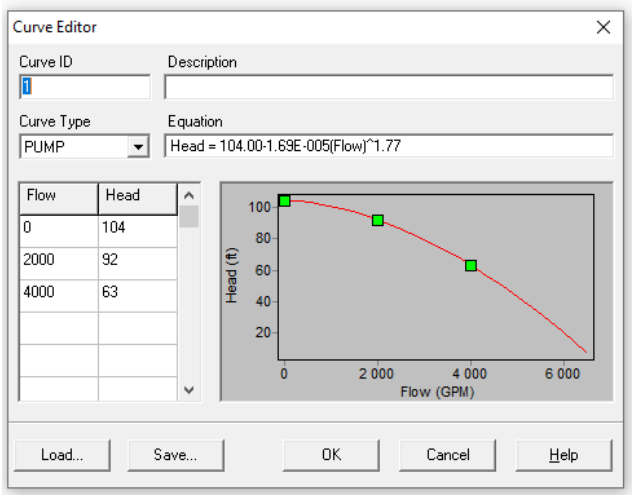
Rys. 4.9 Wynik modelowania przykładowej sieci w programie EPANET [opracowanie własne]

Sieć wodociągowa zamodelowana w powyższy sposób odzwierciedla jedynie strukturę sieci i nie umożliwia jeszcze przeprowadzania symulacji hydraulicznych oraz jakościowych. Aby móc uruchomić symulację bez żadnych błędów konieczne jest jeszcze wprowadzenie informacji o wzorcach rozbioru, krzywych, ustawieniach symulacji oraz jednostek w jakich zostały wprowadzone wszystkie dane.



Rys. 4.10 Przykład wzorca rozbioru w programie EPANET [opracowanie własne]

Aby uzupełnić wzorce rozbioru w modelowanej sieci należy za pośrednictwem ekranu o nazwie *Browser* (zaznaczony kolorem czerwonym na rys. 4.2) wybrać z listy rozwijanej wzorce rozbioru (ang. *Patterns*) i kliknąć *Add*. Spowoduje to otwarcie okna edycji, który umożliwia wpisanie identyfikatora oraz mnożników. Przykład realizacji wzorca rozbioru o identyfikatorze *PAT_1* został przedstawiony na rys. 4.10. Analogicznie do wprowadzania wzorców wprowadza się krzywe pracy pomp (*Browser* > *Curves* > *Add*) oraz definiuje początkowe ustawienia symulacji (*Browser* > *Options*). Na rys. 4.11 przedstawiono przykład wprowadzania krzywej pompy za pomocą metody trzypunktowej.



Rys. 4.11 Przykład wprowadzania krzywej w programie EPANET [opracowanie własne]

W przypadku ustawień podstawowych parametrów symulacji, dostępnych jest pięć zakładek pozwalających na określenie opcji hydraulicznych, jakościowych, reakcji, czasu oraz energii. Parametry hydrauliczne jakie należy ustawić w modelowanym systemie wodociągowym to przede wszystkim [60]:

- jednostka natężenia przepływu (ang. *Flow Units*): GPM;
- formuła obliczania strat liniowych (ang. *Headloss*): H-W;
- maksymalna liczba prób (ang. *Maximum Trials*): 40;
- dokładność obliczeń (ang. *Accuracy*): 0.001
- domyślny wzorzec rozbioru (ang. *Default Pattern*): PAT_1.

Ustawienia początkowe związane z czasem umożliwiają np. ustalenie czasu trwania symulacji (ang. *Total Duration*) czy krok czasowy symulacji hydraulicznej (ang. *Hydraulic Time Step*). O wszystkich możliwościach ustawień w poszczególnych zakładkach można przeczytać więcej w instrukcji obsługi programu EPANET [60].

Przedostatnim etapem modelowania sieci wodociągowej jest wprowadzenie wyrażeń determinujących sposób zarządzania siecią w czasie jej pracy. Wykorzystuje się do tego dwa elementy sterujące: proste i złożone oparte na regułach. Elementy sieci wodociągowej mogą mieć wpływ na właściwości obiektów takich jak poziom wody w zbiorniku czy ciśnienie w analizowanym węźle [60]. Przykładem prostego elementu sterującego może być: *LINK P23 CLOSED IF NODE J15 ABOVE 70* oznaczający przypadek zamknięcia przewodu o identyfikatorze P23 w sytuacji wystąpienia w węźle J15 ciśnienia powyżej 70 stóp [ft]. Złożone elementy sterujące opierają się na formułach umożliwiających powiązanie konkretnych cech obiektów jako reakcji na obecny stan sieci np.: *IF TANK T2 LEVEL BELOW 20.3 THEN PUMP PUMP_1 STATUS IS OPEN AND PIPE P31 STATUS IS OPEN*. Reguła przedstawiona powyżej determinuje uruchomienie pompy o identyfikatorze *PUMP_1*, otwarcie przewodu *P31* w sytuacji gdy, poziom zwierciadła wody w zbiorniku *T2* spadnie poniżej wartości 20.3 stóp [ft]. Szczegółowy opis elementów sterujących oraz ich struktura została omówiona szerzej w pracy [60]. Elementy sterujące niezbędne do zamodelowania pracy przykładowej sieci wodociągowej zostały przedstawione poniżej w postaci listingu:

```
LINK P23 CLOSED IF NODE J15 ABOVE 70
Link P26 CLOSED AT TIME 4
Link P26 OPEN AT TIME 16

IF  TANK T2 LEVEL BELOW 20.3 AND SYSTEM TIME >= 121:00:00
THEN PIPE P31 STATUS IS OPEN ELSE PUMP PUMP_1 STATUS IS OPEN
```

Sieć wodociągowa zamodelowana w taki sposób, zawiera wszystkie niezbędne dane potrzebne do uruchomienia symulacji i przeprowadzenia analizy jej przebiegu. Nie oznacza to jeszcze faktu otrzymywania poprawnych wyników. Dopiero odpowiednio przeprowadzona kalibracja modelu hydraulicznego (ostatni etap, który został szerzej opisany w Rozdziale 4.4) jest gwarancją prawidłowych wyników i poprawnego symulowania procesu dystrybucji wody.

4.3.3 Modelowanie sieci wodociągowej z wykorzystaniem biblioteki WNTR

Pierwszą zasadniczą różnicą w modelowaniu sieci wodociągowej (w odniesieniu do środowiska EPANET) jest brak GUI. Wszystko odbywa się za pośrednictwem odpowiednich metod zaimplementowanych w bibliotece WNTR. Drugim różniącym elementem jest sposób wprowadzania danych oraz jednostki, którymi należy się posługiwać. Wykorzystując WNTR do odzwierciedlania systemu dystrybucji wody (przedstawionego w Rozdziale 4.3.1) należy pamiętać, że wszystkie wartości wejściowe powinny być reprezentowane w jednostkach SI [117]. Oznacza to konieczność konwersji danych w przypadku wystąpienia innych jednostek niż jednostek SI. Dokonać tego można implementując własne funkcje np. funkcja *ft_to_m(value),* która odpowiadałaby za konwersję długości podanej w stopach na metry[2]. W celu stworzenia nowego modelu sieci wodociągowej tworzy się nowy pusty model, który uzupełnia się wykorzystując odpowiednie metody. Poniżej przedstawiono listing kodu umożliwiający utworzenie nowego (pustego) modelu, wraz z zamodelowaniem przykładowych elementów sieci:

```
# ========== Utworzenie nowego pustego modelu:

wn = wntr.network.WaterNetworkModel()

# ========== Modelowanie rezerwuaru:

wn.add_reservoir(R1, base_head=ft_to_m(150),
                     coordinates=(0.00, 80.00))

# ========== Modelowanie zbiornika:

wn.add_tank('T1', elevation=ft_to_m(132.0),
                  init_level=ft_to_m(13.5),
                  min_level=ft_to_m(1),
                  max_level=ft_to_m(32.5),
                  min_vol=0,
                  diameter=ft_to_m(85.0),
                  coordinates=(70.00, 30.00))

# ========== Dodanie nowego wzorca rozbioru:

wn.add_pattern('PAT_1', [0.2, 0.2, 0.6, 0.5, 0.8, 1.2, 1.2, 1.8,
                         2.0, 2.5, 2.5, 3.0, 3.0, 2.5, 2.5, 2.2,
                         2.8, 2.6, 2.3, 2.3, 2.0, 1.5, 1.0, 1.0])
```

[2] Można wykorzystać również gotowe metody konwertujące jednostki dostępne w bibliotece SymPy (https://www.sympy.org/en/index.html)

```
# ========== Modelowanie przykładowego węzła:

wn.add_junction('J5', base_demand=gal_min_to_l_sec(0),
                      demand_pattern='1',
                      demand_category='CAT_3',
                      elevation=ft_to_m(25),
                      coordinates=(5.0, 80.0))

# ========== Modelowanie przewodu:

wn.add_pipe('P1', start_node='R1',
                  end_node='J5',
                  length=ft_to_m(1000),
                  diameter=in_to_mm(12),
                  roughness=100,
                  minor_loss=0.0,
                  status='OPEN')

# ========== Dodanie nowej krzywej:

wn.add_curve('1', curve_type='HEAD',
                  xy_tuples_list=([0.0, 104.0],
                                  [2000.0, 92.0],
                                  [4000.0, 63.0]))

# ========== Modelowanie pompy:

wn.add_pump('PUMP_1', start_node_name='J5',
                      end_node_name='J6',
                      pump_type='HEAD',
                      pump_parameter='1')

# ========== Modelowanie zasuwy:

wn.add_valve('V1', start_node_name='J19',
                   end_node_name='J18',
                   diameter=in_to_mm(18),
                   valve_type='PSV',
                   minor_loss=0,
                   setting=0)
```

Za pośrednictwem powyższych metod można zamodelować całą strukturę sieci wodociągowej. Analogicznie jak w przypadku programu EPANET, pomimo niewielkiego rozmiaru sieci proces modelowania jest czasochłonny i żmudny. Należy na bieżąco analizować i sprawdzać poprawność danych. Nie jest to jednak jedyne rozwiązanie. Wykorzystanie skryptowego podejścia do modelowania sieci wodociągowej umożliwia implementacje generatora modeli, który na podstawie wskazanych baz danych (o określonej strukturze) umożliwi modelowanie sieci

w znacznie szybszym czasie oraz zapewni poprawność wprowadzanych danych (zakładając poprawność danych w bazach danych).

W powyższym przykładzie modelowania sieci wodociągowej z wykorzystaniem skryptów w języku Python i biblioteki WNTR zabrakło elementów sterujących. W programie komputerowym EPANET sterowanie stanem dostępności (otwarcia) przewodów, pompami oraz zaworami odbywa się za pośrednictwem reguł i prostych elementów sterujących (ang. *Controls*). WNTR replikuje funkcjonalność EPANETu oraz wprowadza własne metody. Proste elementy sterujące definiowane są za pośrednictwem warunku *IF* warunek *THEN* akcja. Za pomocą tych elementów możemy wykonywać proste czynności jak otwórz/zamknij przewód, zmień ustawienia w oparciu o określony warunek (np. czas lub poziom cieczy w zbiorniku). Podejście regułowe jest bardziej skomplikowane od prostych elementów sterujących, ponieważ istnieje możliwość implementacji kontr-warunku, w sytuacji niespełnienia się warunku pierwotnego (np. *IF* warunek *THEN* akcja *ELSE* akcja 2). Reguły mogą używać wielu instrukcji warunkowych oraz można wprowadzić ich priorytetyzację. W celu zdefiniowania prostego elementu sterowania lub reguły w bibliotece WNTR należy [117]:

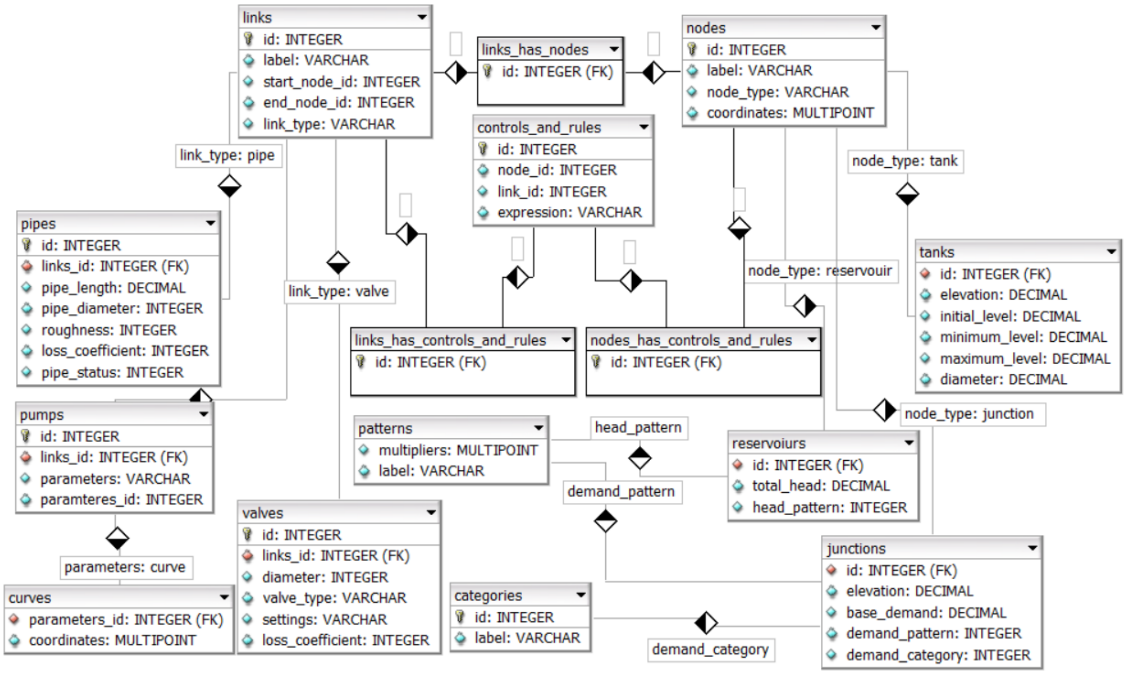
- zdefiniować akcję (czynność, która ma się wykonać w momencie spełnienia pewnego warunku, np. otwarcie przewodu);
- zdefiniować warunki (sytuację, która spowoduje wywołanie akcji, np. poziom cieczy w zbiorniku powyżej 12 metrów);
- zdefiniować prosty element sterowania lub regułę za pośrednictwem połączenia zdefiniowanej wcześniej akcji i warunku;
- dodać utworzony prosty element sterowania lub utworzoną regułę do modelu sieci dystrybucji wody.

Przykład definiowania prostego elementu sterującego oraz reguły został przedstawiony poniżej:

```
# =========== Dodanie prostego elementu sterującego
# =========== LINK P23 CLOSED IF NODE J15 ABOVE 70
# ======== Dodanie akcji:

act1 = controls.ControlAction('P23', 'status', 0)

# ======== Dodanie warunku:

cond1 = controls.ValueCondition('J15', pressure, '<=', 70.0)

# ======== Utworzenie prostego elementu sterowania i dodanie
# ======== go do modelu:

ctr1 = controls.Control(cond1, act1)
wn.add_control('control 1', ctr1)

# =========== Dodanie złożonego elementu sterującego
# =========== IF  TANK T2 LEVEL BELOW 20.3 AND SYSTEM
# =========== TIME >= 121:00:00  THEN PIPE P31 STATUS IS
# =========== OPEN ELSE PUMP PUMP_1 STATUS IS OPEN
# ======== Dodanie akcji:

act1 = controls.ControlAction('P31', 'status', 1)
act2 = controls.ControlAction('PUMP_1', 'status', 1)

# ======== Dodanie warunków:

cond1 = controls.ValueCondition('T2', 'level', '<', 20.3)
cond2 = controls.SimTimeCondition(wn,
                                  controls.Comparison.ge,
                                  '121:00:00')
cond3 = controls.AndCondition(cond1, cond2)

# ======== Utworzenie reguły i dodanie jej do modelu:

rule = controls.Rule(cond3, [act1], [act2], priority=3,
                             name='complex_rule')
wn.add_control('Rule 1', rule)
```

Na potrzeby realizacji zadań postawionych w niniejszej pracy powstał generator modeli, który umożliwia pobieranie danych z baz danych o strukturze zaproponowanej w podrozdziale 4.3.1. Wygenerowanie przykładowego modelu sieci wodociągowej w oparciu o generator modeli oraz zrealizowaną bazę danych zajęło 2.47 sekundy potwierdzając korzyści zaproponowanego podejścia. Generator modeli może również

wygenerować model na podstawie pliku tekstowego .INP, ponieważ jest to wbudowana funkcjonalność biblioteki WNTR. Schemat struktury bazy danych wykorzystywany przez generator modeli przedstawiony został na rys. 4.12.



Rys. 4.12 Struktura bazy danych przechowującej informacje o sieciach wodociągowych [opracowanie własne]

4.4 Kalibracja modeli hydraulicznych systemów dystrybucji wody

Modele hydrauliczne sieci wodociągowych są wiodącym narzędziem inżynierskim, wykorzystywanym przez projektantów, operatorów, nadzorców procesu i eksploratorów. Należy pamiętać, że każdy model odzwierciedla rzeczywistą sieć wodociągową tylko w pewnym stopniu (stopień podobieństwa jest zbliżony). Obowiązkiem projektanta modelu sieci i osób odpowiedzialnych za jego wdrożenie jest więc dążenie do jak największego współczynnika podobieństwa. W tym celu wykonuje się tzw. kalibrację, tj. zespół czynności mających na celu dopasowanie wyników działania modelu hydraulicznego do rzeczywistych wartości występujących w sieci. Umiejętna kalibracja przygotowanego modelu hydraulicznego jest podstawą jego wiarygodności [15, 119]. Niestety ze względu na rozbudowaną topologię sieci wodociągowych, dużą liczbę parametrów jest to zadanie bardzo trudne, często czasochłonne, a co za tym idzie kosztowne. Należy wskazać, że trudności tego procesu są źródłem nieprawidłowości

kalibracji modelu dla stanu ustalonego, do których zaliczyć można dodatkowe źródła zakłóceń [118]. Odwzorowany model hydrauliczny procesu dystrybucji wody nie tylko musi posiadać niezbędną wiedzę o statusie elementów wykonawczych (np. pomp, zasuw), ale musi również przewidywać kiedy ten stan ulegnie zmianie. Konieczne jest nie tylko jak najlepsze odwzorowanie zapotrzebowania bazowego w poszczególnych węzłach, ale również posiadanie wiedzy na temat zmiany tego zapotrzebowania z upływem czasu.

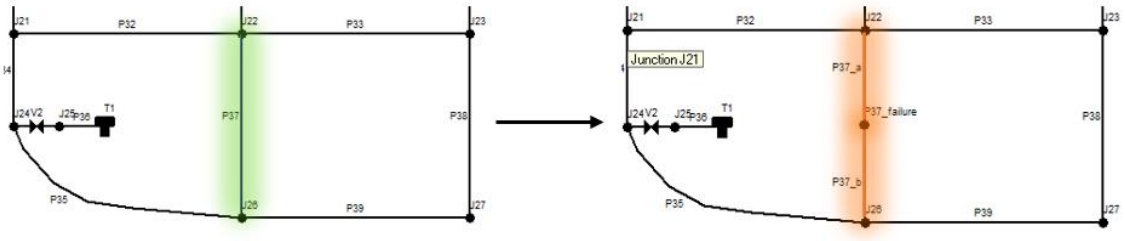
Kalibracji dokonuje się najczęściej na podstawie jednodniowych pomiarów rzeczywistych, tzn. na podstawie danych z sieci wodociągowej (ciśnień, przepływów, itp.) analizuje się symulację hydrauliczną w postaci kroków, porównując i analizując każdy krok w celu polepszenia wiarygodności modelu. Warto pamiętać, że w trakcie kalibracji i porównywaniu wartości zamodelowanych z wartościami rzeczywistymi nie zawsze wina wynikająca z różnicy tych wartości jest po stronie modelu hydraulicznego[3].

4.5 Modelowanie awarii

Jak wspomniano w rozdziale drugim mianem awarii określa się uszkodzenie obiektu lub systemu, wpływające w sposób negatywny na jego dalsze funkcjonowanie przez określony czas. Z punktu widzenia tematyki poruszanej w niniejszej rozprawie modelowanie awarii ma tutaj kluczowe znaczenie. Na przykładzie omawianych narzędzi istnieją dwa sposoby modelowania awarii. Pierwszy sposób, z wykorzystaniem programu komputerowego EPANET, polega na modyfikacji bazowego modelu sieci wodociągowej. W celu zamodelowania przykładowej awarii, wykrytej na przewodzie o identyfikatorze, np. P37, należy zapisać wszystkie parametry przewodu, usunąć go, a następnie dodać nowy węzeł (mniej więcej na środku długości usuniętego przewodu). Po dodaniu nowego węzła tworzy się nowy przewód łączący nowy węzeł z węzłem początkowym przewodu P37 oraz drugi przewód łączący nowo powstały węzeł z węzłem

[3] Przykładem zaprezentowanym w pracy [118] potwierdzającym to stwierdzenie jest kalibracja modelu hydraulicznego sieci wodociągowej w Opolu. Na podstawie pomiarów ciśnień i przepływów z ponad trzydziestu punktów pomiarowych określono stopień korelacji obliczeń hydraulicznych z rzeczywistymi danymi pochodzącymi z urządzeń pomiarowych. Współczynnik korelacji okazał się na tyle wysoki, że często zdarzała się sytuacja, w której obliczenia z modelu były bliższe prawdzie niż dane z punktów pomiarowych. Okazywało się, że kilkanaście punktów pomiarowych zostało błędnie zamontowanych, skonfigurowanych czy było po prostu uszkodzonych. Fakt ten tylko potwierdził wiarygodność zrealizowanego modelu i stanowił podstawę do roszczeń gwarancyjnych i naprawy wadliwych urządzeń pomiarowych.

końcowym przewodu P37. Wynik działania powyższego sposobu został przedstawiony na rys. 4.13.



Rys. 4.13 Modelowanie awarii (wycieku) z wykorzystaniem środowiska EPANET [opracowanie własne]

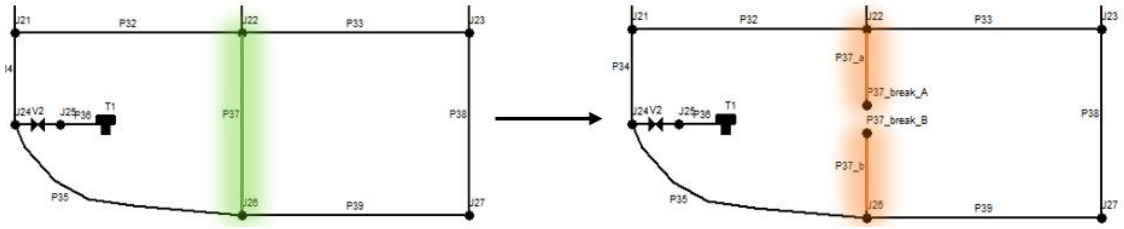
W trakcie modelowania awarii za pomocą sposobu opisanego powyżej należy pamiętać przede wszystkim o poprawności wprowadzanych danych. Suma długości nowo powstałych przewodów powinna być równa długości przewodu usuniętego. Rzędna nowo powstałego węzła powinna być wyliczona na podstawie rzędnej węzła początkowego i końcowego usuniętego przewodu. Należy utworzyć wzorzec symulujący wyciek wody z nowo powstałego węzła i przypisać go do niego. Jak widać metoda modelowania awarii z wykorzystaniem oprogramowania EPANET nie jest może skomplikowana, ale w sytuacji modelowania wielu awarii występujących w jednym modelu staje się bardzo czasochłonna. Biblioteka WNTR umożliwia szybką modyfikację dostępnych modeli sieci wodociągowych wykorzystując język skryptowy Python. Zamodelowanie powyższej sytuacji awaryjnej z wykorzystaniem ww. biblioteki przedstawia się następująco (w poniższym przykładzie zakłada się, że wszystkie dane o usuniętym przewodzie zostały zachowane i przetworzone w taki sposób, aby móc je wykorzystać do zamodelowania awarii):

```
# =========== Usunięcie przewodu o identyfikatorze P37:

wn.remove_link(wn.get_link('P37'))

# =========== Dodanie nowego węzła (symulującego wyciek):

wn.add_junction('P37_failure',
                        base_demand=gal_min_to_l_sec(50),
                        demand_pattern='leak_pattern',
                        demand_category='EN2 base',
                        elevation=ft_to_m(36),
                        coordinates=(85.0, 30.0))
```

```
# ========== Dodanie dwóch nowych przewodów:

wn.add_pipe('P37_a', start_node='J22',
                     end_node='P37_failure',
                     length=ft_to_m(500),
                     diameter=in_to_mm(18),
                     roughness=100,
                     minor_loss=0.0,
                     status='OPEN')

wn.add_pipe('P37_b', start_node='P37_failure',
                     end_node='J26',
                     length=ft_to_m(500),
                     diameter=in_to_mm(18),
                     roughness=100,
                     minor_loss=0.0,
                     status='OPEN')
```

Metoda modelowania awarii zaprezentowana powyżej sprawdza się w przypadku oprogramowania EPANET oraz biblioteki WNTR. O ile w pierwszym przypadku proces dodawania wielu awarii jest czasochłonny (w porównaniu z przypadkiem drugim), o tyle podejście to generuje bardzo dużą ingerencję w model bazowy analizowanego systemu dystrybucji. Usuwanie elementów i zastępowanie go grupą innych powoduje nie tylko zmianę struktury sieci, ale również może wpłynąć na obliczenia hydrauliczne. Taka sytuacja powoduje konieczność przechowywania wielu różnych modeli reprezentujących konkretne przypadki wystąpienia awarii. Zmiana choć jednej awarii w modelu powoduje konieczność zapisania jego kolejnej wersji. Należy również pamiętać, aby przechowywać model bazowy unikając problemu modelowania go na nowo (lub przywracać ustawienia modelu z innego zawierającego awarie).

Modelowanie pęknięć przewodu realizuje się analogicznie jak modelowanie wycieków. Różnica polega na tym, że usuwa się przewód, który uległ pęknięciu (tzw. przewód bazowy), następnie tworzy się dwa nowe węzły (nie jak w przypadku wycieku jeden). Do pierwszego nowego węzła doprowadzany jest nowy przewód, który podłączony jest do węzła początkowego przewodu bazowego, natomiast do drugiego nowego węzła podłącza się drugi nowy przewód, który łączy się z węzłem końcowym przewodu bazowego. Przykład modelowania pęknięcia przewodu (ang. Breaks) został przedstawiony na rys. 4.14.



Rys. 4.14 Modelowanie awarii (pęknięcia) z wykorzystaniem środowiska EPANET [opracowanie własne]

WNTRSimulator umożliwia symulację nieszczelności przewodów wykorzystując równania zaproponowane przez Crowla i Louvara, w którym masowe natężenie przepływu cieczy przez otwór wyraża się jako (4.1) [50, 117]:

$$d_l = C_d \cdot A \cdot P^{\alpha} \sqrt{\frac{2}{\rho}} \tag{4.1}$$

gdzie:

- d_l – oznacza zapotrzebowanie na wodę modelowanego wycieku (m^3/s);
- C_d – oznacza współczynnik rozładowania;
- A – pole przekroju poprzecznego otworu (m^2);
- α – oznacza wykładnik związany z charakterystyką wycieku;
- p – oznacza średni nacisk, nadciśnienie (Pa);
- ρ – oznacza gęstość cieczy (kg/m^3).

Domyślny współczynnik rozładowania wynosi 0.75 (przy założeniu przepływu turbulentnego).W razie potrzeby istnieje możliwość zmiany tego współczynnika. Przykładowo zaleca się zmianę tego współczynnika na 0.5 w sytuacji dużych wycieków z rur stalowych [50]. Nieszczelności (ang. *Leaks*) można dodawać do węzłów i zbiorników. Pęknięcie przewodów modelowane jest poprzez podział rury na dwie części i dodanie węzłów. W sytuacji gdy wykładnik związany z charakterystyką wycieku (α) przyjmie wartość 0.5 równanie 4.1 przyjmuje następującą postać (4.2) [50, 117]:

$$d_l = C_d \cdot A \cdot \sqrt{2 \cdot g \cdot h} \tag{4.2}$$

gdzie:

- d_l – oznacza zapotrzebowanie na wodę modelowanego wycieku (m^3/s);
- C_d – oznacza współczynnik rozładowania;
- A – pole przekroju poprzecznego otworu (m^2);
- h – wysokość (m);
- g – przyspieszenie ziemskie (m/s^2).

Wykorzystując wbudowaną metodę generowania wycieków symulatora WNTR można ponadto ustawić godzinę rozpoczęcia oraz godzinę zakończenia wycieku. Oznacza to brak konieczności przygotowywania kopii modeli z określonymi przypadkami. Wystarczy opracowanie pliku tekstowego, z odgórnie narzuconą strukturą, który będzie wczytywał informację o awariach i generował je w oparciu o wyżej opisaną metodę biblioteki WNTR. Przykład modelowania awarii został przedstawiony poniżej w postaci listingu:

```
# ========== Dodanie wycieku do węzła o identyfikatorze P37
# ========== Metoda 1:

node = wn.get_node('P37')
node.add_leak(wn, area=0.05, start_time=7200, end_time=14400)

# ========== Metoda 2:

wn = wntr.morph.split_pipe(wn, 'P39', 'P39_a', 'leak_sim')
leak_node = wn.get_node('leak_sim')
leak_node.add_leak(wn,area=0.05,start_time=7200,end_time=14400)

# ========== Dodanie pęknięcia przewodu o identyfikatorze P39

wn = wntr.morph.break_pipe(wn,
                    pipe_name_to_split='P39',
                    new_pipe_name='P39_b',
                    new_junction_name_old_pipe='P39_break_A',
                    new_junction_name_new_pipe='P39_break_B',
                    split_at_point=0.5,return_copy='P39_break')

break_node = wn.get_node('P39_break')
breal_node.add_leak(wn, area=0.05,
                        start_time=7200, end_time=14400)
```

Jak można zauważyć, wbudowana metoda biblioteki WNTR znacznie ułatwia sprawę, ponieważ zamodelowanie wycieku/uszkodzenia zajmuje kilka linii kodu. Należy pamiętać, że wycieki w rzeczywistych modelach wodociągowych występują bardzo często i nie są spowodowane wyłącznie sytuacjami katastroficznymi. Mają one miejsce również w wyniku starzejącej się infrastruktury, zmiany warunków atmosferycznych (zamrażanie i rozmrażanie), zwiększonego poboru wody lub nagłych zmian ciśnienia [50, 117].

5. Metodyka typowania zasuw do zamknięcia stosowana w modelowaniu matematycznym systemów wodociągowych

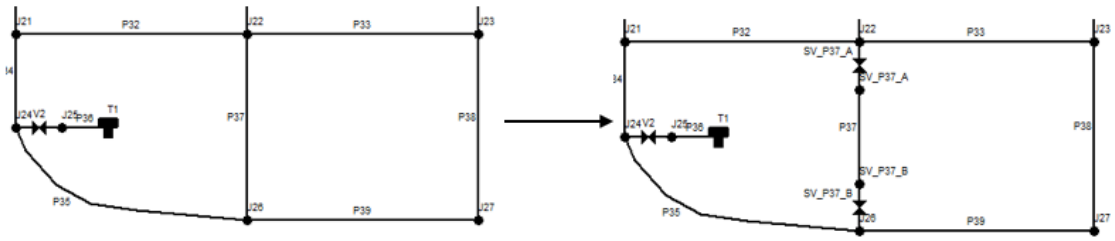
5.1 Wprowadzenie

Kluczową role w sterowaniu SZZwW pełnią zasuwy. Elementy te odpowiadają za kontrolę przepływu, ciśnienia oraz umożliwiają odseparowanie pewnego fragmentu sieci. Umiejętność izolacji podsystemu poprzez zamknięcie określonej liczby zasuw pozwala na wykonanie odpowiednich czynności naprawczych. Czynności te, polegające na naprawie, konserwacji lub wymianie uszkodzonego elementu spowodują przywrócenie sprawności podsystemu, a w konsekwencji całego systemu wodociągowego. Z punktu widzenia eksploatatora odseparowany odcinek sieci, w skład którego wchodzą przewody wodociągowe oraz węzły, nosi miano segmentu (lub zamiennie sektora). Wielkość segmentu determinowana jest poprzez liczbę elementów wchodzących w jego skład, która określana jest na podstawie stanu oraz alokacji poszczególnych zasuw. Wiedza o lokalizacji oraz składzie poszczególnych segmentów powinna być monitorowana i aktualizowana w sposób dynamiczny na podstawie przeglądów technicznych elementów wykonawczych. Niestety obecnie bardzo często proces ten jest pomijany, co przekłada się na czas realizacji poszczególnych zadań naprawczych. Z informacji uzyskanych od nadzorcy procesu dystrybucji wody, nadzór sprawności zasuw umożliwi uniknięcie częstej sytuacji, w której brygada naprawcza po dotarciu na miejsce występowania zasuwy dowiaduje się, że jej stan nie pozwala na odseparowanie danego odcinka. W takiej sytuacji ekipa realizująca zadanie naprawcze, na podstawie danych o strukturze sieci wodociągowej oraz lokalizacji zasuw, określa zbiór dodatkowych zasuw umożliwiających izolację danego segmentu lub grupy segmentów.

W dalszej części dokonano przeglądu literatury na temat rozwiązań informatycznych, które w sposób szczególny mogą przyczyniać się do usprawnienia procesu typowania zasuw niezbędnych do odseparowania segmentu, w którym wystąpiła jedna lub kilka awarii. W rozdziale przedstawiono również wybraną metodę typowania zasuw do zamknięcia wraz z jej niewielką modyfikacją.

5.2 Sposoby modelowania zasuw odcinających

Zasuwy odcinające (ang. *Shutoff Valve*) oraz zasuwy zwrotne (ang. *Non-return Valve*), których zadaniem jest całkowite otwarcie lub zamknięcie przewodu nie są traktowane jako oddzielne elementy w środowisku EPANET. Powyższe urządzenia wykonawcze uwzględnione są jako właściwości przewodu, w którym są umieszczone. W celu odseparowania wskazanego odcinka w systemie dystrybucji wody należy zmienić w opcji przewodu atrybut *Status* na wartość *Close*. Jest to przydatna opcja umożliwiająca analizę pracy sieci wodociągowej w sytuacji izolacji poszczególnych odcinków. Minusem tego rozwiązania jest jednak brak informacji o stanie poszczególnych zasuw (ponieważ EPANET zakłada, że wszystkie zasuwy można zamknąć) oraz o ich lokalizacji, co z punktu widzenia niniejszej pracy jest bardzo istotne. Postrzeganie zasuw odcinających w środowisku EPANET na przykładzie przewodu o identyfikatorze *P37* zostało przedstawione na rys. 5.1.



Rys. 5.1 Sposób interpretacji zasuw odcinających w środowisku EPANET [opracowanie własne]

W przykładzie opisanym powyżej (rys. 5.1), zakłada się obecność na każdym przewodzie dwóch zasuw odcinających, znajdujących się blisko początku i końca odcinka (identyfikatory: *SV_P37_A* oraz *SV_P37_B*). W sytuacji konieczności jego izolacji (zmiany atrybutu *Status*) zakłada się, że obie zasuwy zostają zamknięte.

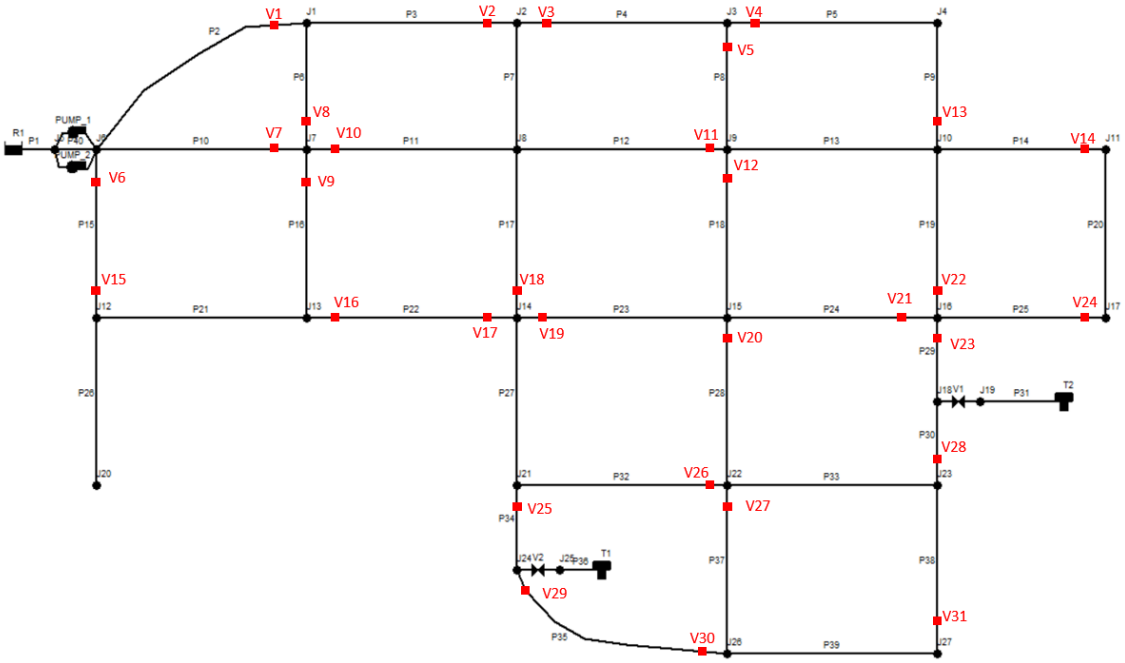
W systemach zbiorowego zaopatrzenia w wodę bardzo rzadko można spotkać się z sytuacją, w której każdy przewód wyposażony jest w dwie zasuwy odcinające. Taka sytuacja spowodowana jest kosztami przedsięwzięcia. W trakcie projektowania sieci na podstawie opracowanej topologii wyznacza się odpowiednią lokalizację tego typu

elementów wykonawczych czy punktów pomiarowych, aby jak najlepiej zarządzać procesem dystrybucji wody.

Przykład badań opisujących algorytmy alokacji punktów monitorowania jakości w systemach wodociągowych został opisany w pracy doktorskiej [52] oraz artykułach [34, 51, 82, 104]. Opisują one działanie algorytmów rozmieszczania pewnej wystarczającej liczby urządzeń pomiarowych jakości wody w taki sposób, aby na podstawie wartości pomiarów z tych punktów otrzymać właściwą wiedzę na temat stanu jakości wody w całym systemie dystrybucji wody. Wyniki tych badań wykorzystać można również w problematyce lokalizacji zasuw odcinających, których odpowiednie rozmieszczenie pozwoli w sposób skuteczny zarządzać dystrybucją wody w warunkach normalnych i kryzysowych. W pracach [17, 58] przedstawiono nowe zastosowanie programowania logicznego CLP-FD (ang. *Constraint Logic Programming over Finite Domains*) rozwiązującego problem lokalizacji optymalnej liczby zasuw odcinających. Wartością dodaną tego artykułu jest współpraca wielu naukowców specjalizujących się w hydro informatyce, inżynierii środowiska czy automatyce. Różnorodność zespołu pod względem wykształcenia pozwoliła na lepsze zrozumienie i postrzeganie procesu z punktu widzenia różnych dyscyplin naukowych. Inne rozwiązania zaproponowano w artykułach [23, 27], które wykorzystują wielozadaniowe algorytmy metaheurystyczne. W zastosowanych podejściach wykorzystano i porównano różne funkcje celu, chcąc znaleźć optymalne rozwiązanie pozycji określonej liczby zasuw. Przykładową funkcją celu analizowaną ww. artykule była minimalizacja całkowitego kosztu zasuw oraz minimalizacja średniego niedoboru wody. W przypadku artykułu [23], rozwiązanie zostało przetestowane na modelu matematycznym rzeczywistego systemu dystrybucji wody w mieście Ferrara we Włoszech. Artykuły [30, 31] przedstawiają nowatorską metodologię oceny systemu zasuw odcinających i poszczególnych segmentów sieci wodociągowej, które zostały zaizolowane przez zamknięcie tych zasuw. Zaproponowane rozwiązanie opisuje algorytm identyfikacji zależności między zasuwami a odseparowanymi segmentami. Podejście to wykorzystuje macierze topologiczne sieci, której topologia modyfikowana jest w celu uwzględnienia istnienia zbioru zasuw.

W celu poprawnego zarządzania systemem dystrybucji wody konieczna jest wiedza na temat stanu i lokalizacji zasuw odcinających. O ile środowisko EPANET nie

umożliwia modelowania poszczególnych zasuw o tyle biblioteka WNTR już tak. Pomimo, że zasuwy odcinające nie są modelowane jako osobne elementy infrastruktury sieci wodociągowej, wykorzystywana biblioteka umożliwia zdefiniowanie warstwy zasuw (ang. *Valve Layer*). Warstwa ta może zostać wykorzystana do szerszej analizy przykładowej sieci wodociągowej oraz grupowania przewodów i węzłów w segmenty (bazując na lokalizacji zasuw odcinających). W warstwie zasuw, ich lokalizacja definiowana jest za pośrednictwem identyfikatora przewodu, na którym się znajduje oraz identyfikatora węzła, do którego przylega. Informacje te przechowywane są w strukturze tabelarycznej [117]. W celu analizy poprawności działania wszystkich zaproponowanych rozwiązań przykładowa sieć (rys. 4.9) została wyposażona w losowo rozmieszczone zasuwy odcinające. Przykładowe rozmieszczenie zasuw odcinających przedstawiono na rys. 5.2.



Rys. 5.2 Przykładowy model sieci wodociągowej z lokalizacją zasuw odcinających [opracowanie własne]

Aby wygenerować losową konfigurację rozmieszczenia określonej liczby zasuw w przykładowym systemie zbiorowego zaopatrzenia w wodę wykorzystano wbudowaną funkcję biblioteki WNTR. Poniższa instrukcja przedstawia użycie metody

generate_valve_layer, która w sposób losowy rozmieściła w systemie wodociągowym trzydzieści jeden zasuw odcinających:

```
valve = wntr.network.generate_valve_layer(self.wn, 'random', 31)
```

W Tabeli 5.1 przedstawiono szczegółową lokalizację zasuw odcinających. Zgodnie ze specyfikacją poszczególne zasuwy definiowane są na podstawie identyfikatora przewodu oraz węzła. W celu określenia dokładniejszych współrzędnych założono, że w przykładowym modelu zasuwa znajduje się odległości dwóch jednostek od węzła. Przykładowo zasuwa o identyfikatorze *V1*, znajdująca się na przewodzie *P2* oraz przy węźle *J1* o współrzędnych (35, 95) znajduje się w miejscu o współrzędnych (33, 95).

Tab. 5.1 *Informacje o zasuwach odcinających w modelowanym systemie [opracowanie własne]*

Informacje o zasuwach odcinających (typ: Shutoff Valve)								
Identyfikator [ID]	**Węzeł** [ID]	**Przewód** [ID]	**Współrzędne**		**Identyfikator** [ID]	**Węzeł** [ID]	**Przewód** [ID]	**Współrzędne**
V1	J1	P2	(33.0, 95.0)		V17	J14	P22	(58.0, 60.0)
V2	J2	P3	(58.0, 95.0)		V18	J14	P17	(60.0, 62.0)
V3	J2	P4	(62.0, 95.0)		V19	J14	P23	(62.0, 60.0)
V4	J3	P5	(87.0, 95.0)		V20	J15	P28	(85.0, 58.0)
V5	J3	P8	(85.0, 93.0)		V21	J16	P24	(108.0, 60.0)
V6	J6	P15	(10.0, 82.0)		V22	J16	P19	(110.0, 62.0)
V7	J7	P10	(33.0, 80.0)		V23	J16	P29	(110.0, 58.0)
V8	J7	P6	(35.0, 82.0)		V24	J17	P25	(128.0, 60.0)
V9	J7	P16	(35.0, 78.0)		V25	J21	P34	(60.0, 38.0)
V10	J7	P11	(37.0, 80.0)		V26	J22	P32	(83.0, 40.0)
V11	J9	P12	(83.0, 80.0)		V27	J22	P37	(80.0, 38.0)
V12	J9	P18	(80.0, 82.0)		V28	J23	P30	(110.0, 42.0)
V13	J10	P9	(110.0, 82.0)		V29	J24	P35	(60.0, 28.0)
V14	J11	P14	(128.0, 80.0)		V30	J26	P35	(83.0, 20.0)
V15	J12	P15	(10.0, 62.0)		V31	J27	P38	(110.0, 22.0)
V16	J13	P22	(37.0, 60.0)		-	-	-	-

5.3 Algorytm typowania zasuw do zamknięcia

Algorytm wyszukiwania segmentów przedstawiony w poniższym Rozdziale (5.3.1) został opracowany na podstawie zbioru macierzy i stanowi efekt pracy doktorskiej pt. *Strategic valve locations in a water distribution system* autorstwa Hwadon Jun [35].

5.3.1 Podejście macierzowo-grafowe

W oparciu o strukturę analizowanego systemu zbiorowego zaopatrzenia w wodę przedstawionego na rys. 5.2 zostają utworzone dwie macierze. W pierwszej macierzy (Macierz A) kolumny reprezentowane są za pośrednictwem identyfikatorów przewodów, natomiast wiersze przedstawiają identyfikatory węzłów wchodzących w skład sieci. Na podstawie topologii sieci, macierz uzupełniana jest informacją o zależności między poszczególnymi elementami. Przykładowo jeśli w analizowanej sieci przewód o identyfikatorze *P2*, koreluje[4] z węzłem o identyfikatorze *J1*, to w macierzy o współrzędnych (P2, J1) wpisywana zostaje wartość logiczna 1.

Tab. 5.2 (M*acierz A) analizowanego przykładu SDW [opracowanie własne]*

Macierz A – przykładowy system dystrybucji wody

-	P1	P10	P11	P12	P13	P14	P20	P25	P19	P24	P28	P33	P38	P39	P37	P32	P34	P35	P36	P26	P15	P21	P22	P23	P27	P17	P16	P6	P7	P8	P18	P9	P5	P4	P3	P2	P29	P30	P31	P40	PUMP_1	PUMP_2	V2	V1
J5	1																																							1	1	1		
R1	1																																												
J6		1																			1															1				1	1	1		
J7		1	1																								1	1																
J8			1	1																						1			1															
J9				1	1																									1	1													
J10					1	1			1																							1												
J1																											1							1	1									
J2																												1					1	1									
J3																														1			1	1									
J4																																1	1										
J13																						1	1				1																	
J14																							1	1	1	1																		
J15										1	1													1							1													
J21																1	1								1																			
J22											1	1			1	1																												
J23												1	1																											1				
J16								1	1	1																																		
J12																				1	1	1																						
J24																	1	1																									1	
J25																			1																								1	
J11						1	1																																				
J17							1	1																																				
J26														1	1			1																										
J27													1	1																														
J20																				1																								
J18																																						1	1					1
J19																																							1					1
T1																			1																									
T2																																							1					

Tabela 5.2 przedstawia (Macierz A) analizowanego przykładowego systemu zbiorowego zaopatrzenia w wodę. Druga macierz (Macierz B) zawiera informacje na temat lokalizacji zasuw. Wiersze i kolumny reprezentowane są analogicznie jak w macierzy pierwszej poprzez identyfikatory węzłów oraz przewodów. W tym przypadku natomiast wartość logiczna 1 wpisywana jest do macierzy o określonych współrzędnych

[4] Poprzez korelację, rozumie się, że węzeł *J1* jest węzłem początkowym lub końcowym przewodu *P2.*

wtedy i tylko wtedy, gdy przy danym węźle, na określonym przewodzie zlokalizowana jest zasuwa. Tabela 5.3 przedstawia (Macierz B) analizowanego przykładowego systemu wodociągowego.

Tab. 5.3 (M*acierz B) analizowanego przykładu SDW [opracowanie własne]*

Macierz B – przykładowy system dystrybucji wody

-	P1	P10	P11	P12	P13	P14	P20	P25	P19	P24	P28	P33	P38	P39	P37	P32	P34	P35	P36	P26	P15	P21	P22	P23	P27	P17	P16	P6	P7	P8	P18	P9	P5	P4	P3	P2	P29	P30	P31	P40	PUMP_1	PUMP_2	V2	V1
J5																																												
R1																																												
J6																					1																								
J7		1	1																								1	1																
J8																																												
J9				1																											1												
J10																																1												
J1																																			1									
J2																																		1	1									
J3																														1			1											
J4																																												
J13																							1																						
J14																							1	1		1																			
J15											1																																	
J21																	1																												
J22															1	1																													
J23																																						1						
J16									1	1																												1						
J12																					1																								
J24																		1																											
J25																																												
J11						1																																					
J17								1																																			
J26																		1																											
J27													1																																
J20																																												
J18																																												
J19																																												
T1																																												
T2																																												

Na podstawie dwóch wygenerowanych macierzy, zostaje wytworzona trzecia macierz (Macierz C), która określa segmenty w analizowanym modelu hydraulicznym sieci wodociągowej. (Macierz C) powstaje w wyniku różnicy (Macierz A) oraz (Macierz B) tj. (Macierz C) = (Macierz A) – (Macierz B). Tabela 5.4 przedstawia (Macierz C) analizowanego przykładowego systemu wodociągowego.

Tab. 5.4 (M*acierz C) analizowanego przykładu SDW [opracowanie własne]*

Macierz C – przykładowy system dystrybucji wody

-	P1	P10	P11	P12	P13	P14	P20	P25	P19	P24	P28	P33	P38	P39	P37	P32	P34	P35	P36	P26	P15	P21	P22	P23	P27	P17	P16	P6	P7	P8	P18	P9	P5	P4	P3	P2	P29	P30	P31	P40	PUMP_1	PUMP_2	V2	V1
J5	1																																							1	1	1		
R1	1																																												
J6		1																																		1				1	1	1		
J7																																												
J8			1	1																					1			1															
J9					1																								1														
J10					1	1			1																																			
J1																												1						1									
J2																													1														
J3																																	1									
J4																																1	1										
J13																						1					1																	
J14																									1																			
J15										1														1							1													
J21																1									1																			
J22											1	1																															
J23												1	1																															
J16								1																																			
J12																				1		1																						
J24																	1																										1	
J25																			1																								1	
J11							1																																				
J17							1																																				
J26														1	1																													
J27														1																														
J20																				1																								
J18																																					1	1						1
J19																																							1					1
T1																			1																									
T2																																							1					

W oparciu o (Macierz C) wyznaczyć można przynależność wybranych elementów do określonych segmentów. Przykładowo, została zgłoszona awaria w przewodzie o identyfikatorze *P38*. Macierz pozwoli określić, które zasuwy należy zamknąć aby odizolować uszkodzony przewód i podjąć się jego naprawy. Pierwszy etap algorytmu typowania zasuw do zamknięcia przebiega w sposób następujący [35]:

(1) Utworzone zostają dwie puste listy o nazwach *pipe_list* oraz *node_list*, których zadaniem jest przechowywanie informacji o przewodach i węzłach wchodzących w skład analizowanego segmentu.

```
# == Krok (1): Stan list po zakończeniu iteracji (1)
pipe_list = []
node_list = []
```

(2) Dodanie do listy *pipe_list* identyfikatora przewodu, w którym wystąpiła awaria.

```
# == Krok (2): Stan list po zakończeniu iteracji (2)
pipe_list = [P38]
node_list = []
```

(3) Wyszukiwanie w (Macierzy C) w kolumnie o identyfikatorze *P38* wartości logicznej 1. Algorytm odnalazł wskazaną wartość w wierszu *J23*, którego identyfikator zostaje dodany do listy *node_list*. W związku z odnalezieniem wartości 1 przejdź do kroku 4.

```
# == Krok (3): Stan list po zakończeniu iteracji (3)
pipe_list = [P38]
node_list = [J23]
```

(4) Wyszukiwanie wartości 1 w wierszu o identyfikatorze *J23*. Algorytm odnalazł szukaną wartość w kolumnie *P33*, której identyfikator zostaje dodany do odpowiedniej listy.

```
# == Krok (4): Stan list po zakończeniu iteracji (4)
pipe_list = [P38, P33]
node_list = [J23]
```

(3) W związku z odnalezieniem poszukiwanej wartości algorytm wraca do kroku trzeciego, w którym następuje ponowne przeszukiwanie kolumny, tym razem o identyfikatorze *P33*. Algorytm odszukał wartość w wierszu o identyfikatorze *J22*, który został dodany do listy *node_list*.

```
# == Krok (3): Stan list po zakończeniu iteracji (5)
pipe_list = [P38, P33]
node_list = [J23, J22]
```

(4) W związku z odnalezieniem poszukiwanej wartości algorytm ponownie rozpoczyna krok 4, w którym następuje ponowne przeszukiwanie wierszy. W wierszu *J22* algorytm odnalazł wartość 1 w kolumnie *P28*, której identyfikator trafia do listy *pipe_list*. Algorytm ponownie wraca do kroku 3.

```
# == Krok (4): Stan list po zakończeniu iteracji (6)
pipe_list = [P38, P33, P28]
node_list = [J23, J22]
```

(3) W trakcie przeszukiwania kolumny *P28* algorytm nie odnalazł żadnej wartości 1. Oznacza to, że należy przejść do kroku 5 i zakończyć swoje zadanie.

(5) Zakończ działanie.

Wynikiem pierwszego etapu działania algorytmu jest lista elementów tworzących tzw. segment. W analizowanym przypadku są to węzły o identyfikatorach *J22* i *J23* oraz przewody o identyfikatorach *P28, P33* oraz *P38*. Warto dodać, że w sytuacji wystąpienia w kolumnie lub wierszu większej liczby wartości 1, dla każdego przypadku trzeba dokonać osobnej analizy [35].

Tab. 5.5 *Prezentacja działania pierwszego etapu algorytmu (Macierz C) [opracowanie własne]*

Macierz C – Prezentacja działania pierwszego etapu algorytmu

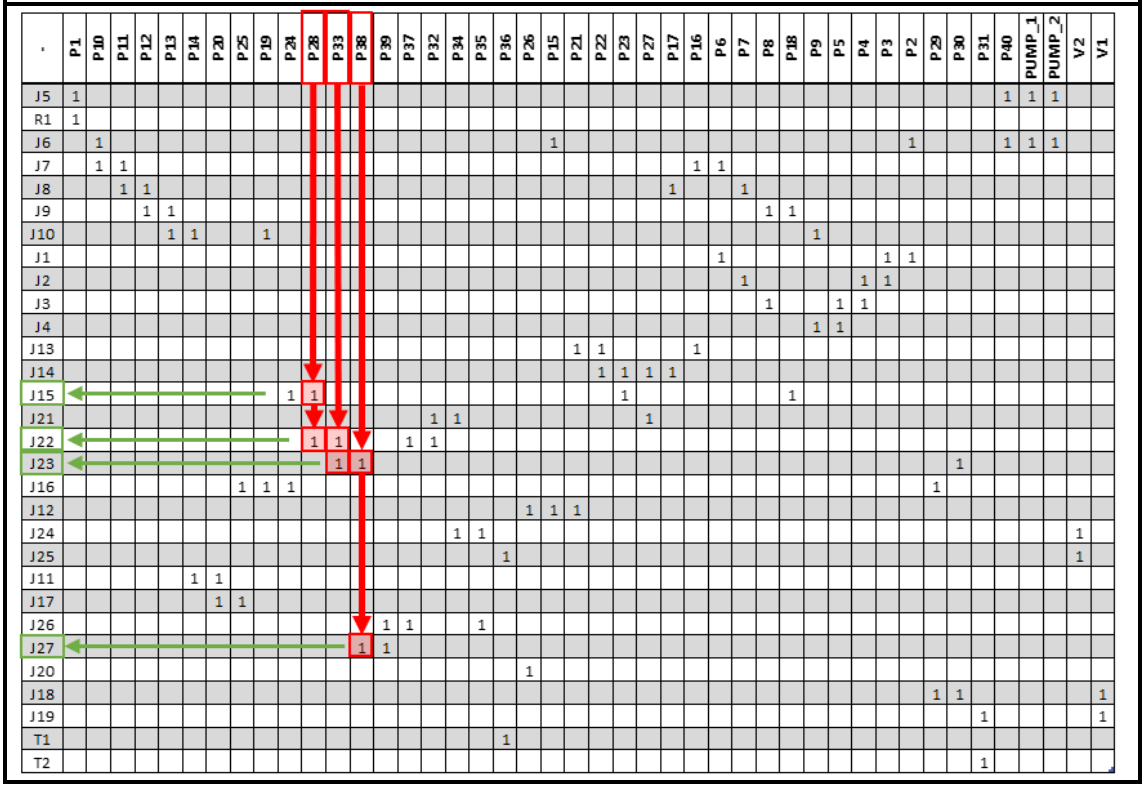
·	P1	P10	P11	P12	P13	P14	P20	P25	P19	P24	P28	P33	P38	P39	P37	P32	P34	P35	P36	P26	P15	P21	P22	P23	P27	P17	P16	P6	P7	P8	P18	P9	P5	P4	P3	P2	P29	P30	P31	P40	PUMP_1	PUMP_2	V2	V1
J5	1																																							1	1	1		
R1	1																																												
J6		1																																		1				1	1	1		
J7																																												
J8			1	1																						1			1															
J9					1					3			1																	1														
J10					1	1			1																																			
J1																												1							1								
J2																													1														
J3																																		1									
J4																																1	1										
J13																						1				1																		
J14																									1																			
J15					4				1															1							1													
J21																1									1																			
J22											1	1																																
J23												1	1																															
J16					2			1																																				
J12																				1		1																						
J24																	1																										1	
J25																			1																								1	
J11							1																																				
J17							1																																				
J26														1	1																													
J27														1																													
J20																				1																							
J18																																				1	1						1
J19																																							1				1
T1																			1																								
T2																																							1				

W Tabeli 5.5 przedstawiono krokową prezentację działania pierwszego etapu algorytmu, polegającej na przeszukiwaniu (Macierzy C) w celu określenia zbioru elementów niezbędnych do odizolowania. Na podstawie wyznaczonego segmentu (oznaczonego identyfikatorem *S15*) algorytm rozpoczyna drugi etap swojego działania, którego zadaniem jest wyznaczenie zasuw niezbędnych do zamknięcia segmentu. Aby zlokalizować węzły znajdujące się w pobliżu wyznaczonego segmentu, należy w oparciu o (Macierz A) odszukać wszystkie identyfikatory węzłów, które korelują z przewodami znajdującymi się w liście *pipe_list*. Nowe węzły należy dodać do nowo powstałej listy

near_node_list. W Tabeli 5.6 przedstawiono krokową prezentację działania drugiego etapu algorytmu typowania zasuw do zamknięcia.

Tab. 5.6 Prezentacja działania drugiego etapu algorytmu (Macierz A) *[opracowanie własne]*

Macierz A – Prezentacja działania pierwszego drugiego algorytmu



Po zakończeniu działania drugiego etapu algorytmu, lista *near_node_list* przechowuje informacje o zasuwach, które znajdują się w pobliżu analizowanego segmentu. W rozpatrywanym przykładzie etap drugi do nowo powstałej listy dodał dwa identyfikatory: *J15* oraz *J27*. Warto tutaj nadmienić, że do nowej listy trafiają identyfikatory tylko tych węzłów, które nie znajdują się w liście *node_list*. Poniżej została przedstawiona aktualna wartość list *pipe_list*, *node_list* oraz *near_node_list*.

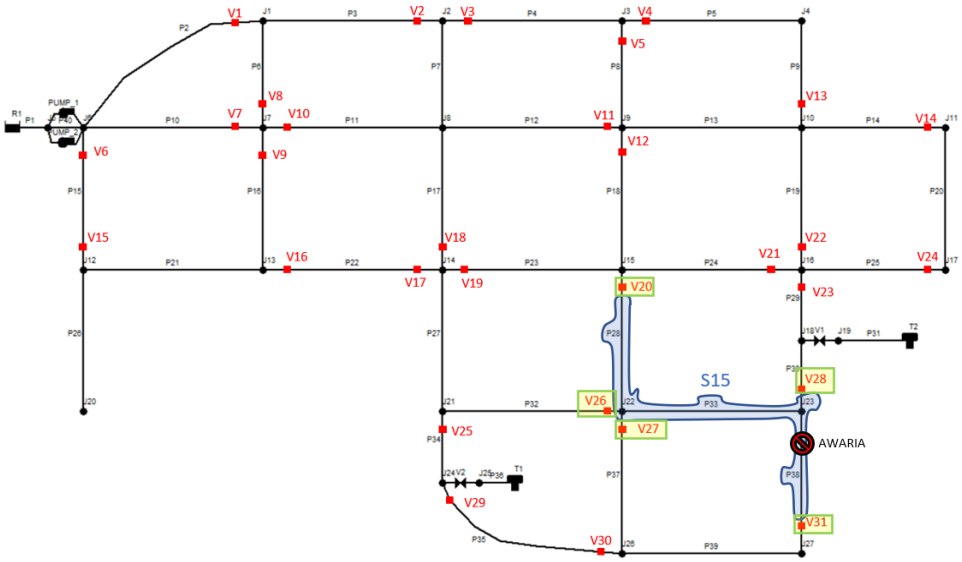
```
# == Listy przewodów i węzłów po zakończeniu 2 etapu algorytmu
pipe_list = [P38, P33, P28]
node_list = [J23, J22]
near_node_list = [J15, J27]
```

Trzecim etapem algorytmu jest określenie identyfikatorów zasuw znajdujących się na przewodach wchodzących w skład segmentu S15 lub znajdujących się w pobliżu

węzłów, których identyfikatory znajdują się w listach *node_list* oraz *near_node_list*. W celu określenia zbioru identyfikatorów zasuw analizie podlega baza danych przechowująca informacje o zasuwach (Tabela 5.1) oraz wynikowe listy etapu drugiego algorytmu. W analizowanym przykładzie wyznaczono następujący zbiór zasuw (pogrubione identyfikatory prezentują wartości znajdujące się w poszczególnych listach wynikowych drugiego etapu algorytmu):

- V20 – znajdujący się przy przewodzie ***P28*** oraz węźle ***J15****;*
- V26 – znajdujący się przy przewodzie *P32* oraz węźle ***J22***;
- V27 – znajdujący się przy przewodzie *P37* oraz węźle ***J22***;
- V28 – znajdujący się przy przewodzie *P30* oraz węźle ***J23***;
- V31 – znajdujący się przy przewodzie ***P38*** oraz węźle ***J27***.

W przypadku przewodu o identyfikatorze *P33* nie znaleziono identyfikatora żadnej zasuwy (oznacza to brak zasuw na analizowanym przewodzie). Wynikiem ostatniego etapu (trzeciego) algorytmu jest zbiór zasuw (*V20, V26, V27, V28, V31*), koniecznych do odizolowania analizowanego segmentu. Na rys. 5.3 przedstawiono powstały segment w postaci graficznej z zaznaczonymi zasuwami, które należy zamknąć w celu dokonania czynności naprawczych zgłoszonej awarii (przewód *P38*).



Rys. 5.3 Przykładowy segment (S15) w analizowanym przypadku sieci wodociągowej [opracowanie własne]

5.3.2 Podejście macierzowo-grafowe z analizą przepływów

Rozwiązanie przedstawione w Rozdziale 5.3.1 stanowi kluczową bazę rozwojową algorytmu, którego zadaniem jest wyznaczenie minimalnego zbioru zasuw niezbędnych do odizolowania wskazanego odcinka. Na podstawie analizy symulacji hydraulicznych nie zawsze konieczne jest zamykanie wszystkich zasuw. Okazuje się, że zamknięcie już części z nich powoduje brak wody w przewodzie, który ma zostać naprawiony. Sytuacja ta najczęściej spowodowana jest lokalizacją uszkodzonego przewodu (np. końcówka sieci), ukształtowaniem terenu, niskim ciśnieniem lub średnicą przewodu. Czas potrzebny na dojazd ekipy naprawczej, lokalizację zasuwy oraz sam proces jej zamknięcia jest długi, co w sytuacji kryzysowej (wielu awarii) ma ogromne znaczenie. W celu poprawy efektywności procesu wyznaczania zasuw do zamknięcia zaproponowano zmodyfikowaną metodę macierzowo-grafową opartą na symulacji hydraulicznej modelu sieci wodociągowej i analizie przepływów.

Na podstawie wyznaczonego zbioru zasuw (*V20, V26, V27, V28*, *V31*) zostaje wygenerowany pierwszy scenariusz symulacyjny. Scenariusz wygenerowany jest w postaci pliku tekstowego, którego struktura została przedstawiona poniżej:

```
#==== Scenario stage (1) | Net: AA_pdh.inp | Use case: 1
(P) P38
(1) V20,V26,V27,V28
(2) V20,V26,V27,V31
(3) V20,V26,V28,V31
(4) V20,V27,V28,V31
(5) V26,V27,V28,V31
```

W pierwszej linii umieszczany jest komentarz, który służy do identyfikacji scenariusza. Znajdują się tam informacje takie jak etap scenariusza, nazwa pliku, w którym znajduje się model poddawany analizie oraz numer przypadku testowego. W drugiej linii znajduje się lista przewodów, w których należy sprawdzać przepływ (ang. *Flow*). Kolejne linie pliku zawierają przypadki testowe, które należy przeanalizować. Powyższy scenariusz przewiduje więc ewaluację modelu o nazwie *AA_phd.inp*, w którym w przewodzie o identyfikatorze *P38* będzie sprawdzany przepływ. Analiza przepływu w uszkodzonym odcinku będzie bazowała na pięciu symulacjach hydraulicznych, w których zostaną zamknięte zasuwy zgodne z listą (1) – (5). Kontrola przepływu polega

na sprawdzaniu jego wartości w ciągu trwania całej symulacji. Jeśli we wszystkich krokach symulacji, jego wartość jest mniejsza lub równa zero, oznacza to, że analizowany zbiór zasuw jest wystarczający do jego odizolowania. Symulacje hydrauliczne poszczególnych przypadków testowych uruchamiane są z wykorzystaniem biblioteki WNTR i zakładają 24 godzinną pracę systemu wodociągowego. Warto tutaj nadmienić, że skrypt ten został oparty o bibliotekę Ray[5], która umożliwia wykorzystanie wielowątkowości. Oznacza to, że wszystkie symulacje odbywały się w tym samym czasie, minimalizując czas oczekiwania na wynik. Rezultatem zaimplementowanego skryptu są dwa nowe pliki z dopiskami „_result" oraz „_report".

W przypadku raportu („_report") plik zawiera szczegółowe informację o przebiegu wszystkich symulacji, natomiast plik wynikowy („_result") zawiera zmodyfikowany plik scenariusza z krótką informacją T (True) lub F (False). Na jej podstawie można szybko wywnioskować czy poszczególny zbiór zasuw wystarczy do odizolowania wskazanego odcinka. Poniżej przedstawiono zawartość pliku wynikowego:

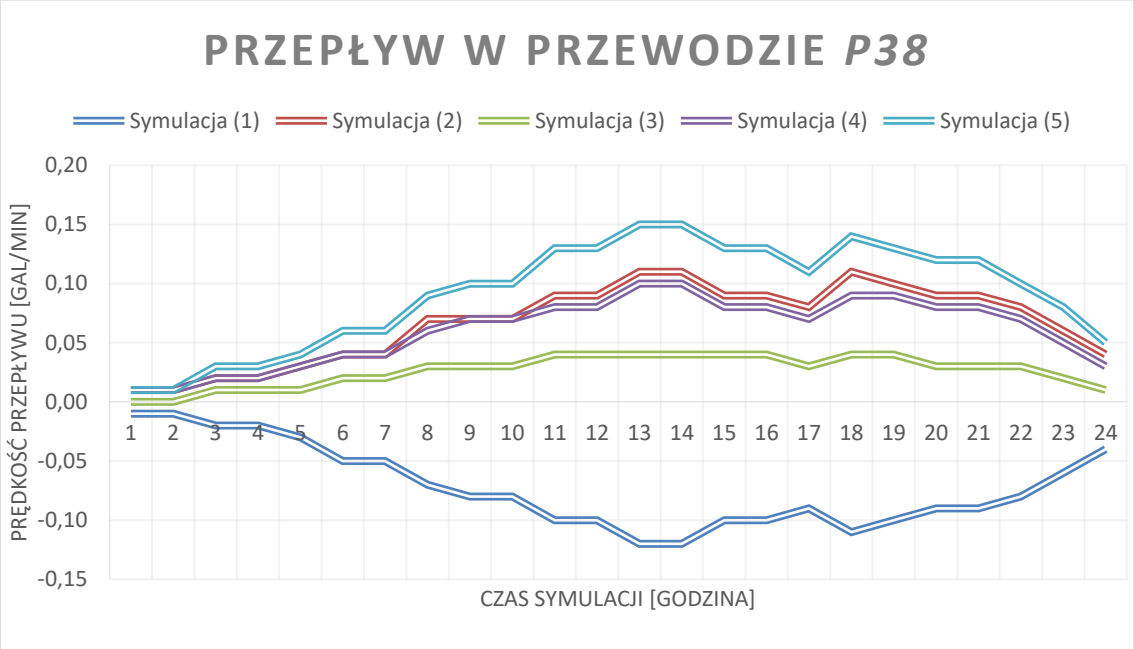
```
#==== Scenario stage (1) | Net: AA_pdh.inp | Use case: 1
(P) P38
(1) V20,V26,V27,V28 T
(2) V20,V26,V27,V31 F
(3) V20,V26,V28,V31 F
(4) V20,V27,V28,V31 F
(5) V26,V27,V28,V31 F
```

W oparciu o wyniki symulacji hydraulicznej okazuje się, że w przypadku analizowanej awarii przewodu o identyfikatorze *P38* ze wskazanego zbioru pięciu zasuw (*V20, V26, V27, V28, V31*) należy zamknąć tylko cztery z nich. Zamknięcie zasuw (*V20, V26, V27, V28*) jest wystarczającym zbiorem potrzebnym do odizolowania przewodu o identyfikatorze *P38*.

Na rys. 5.4 przedstawiono analizę przepływu w przewodzie *P38* w poszczególnych symulacjach. W przypadku symulacji hydraulicznych (2), (3), (4) oraz (5) zauważyć można zbliżone wyniki. Oznacza to, że zasuwy o identyfikatorze: *V20, V26, V27, V28* są niezbędne w sytuacji konieczności odizolowania uszkodzonego przewodu. Wartości

[5] Więcej o bibliotece Ray: https://www.ray.io/

przepływu w symulacji (1) jednoznacznie wskazują na to, że zasuwa o identyfikatorze *V31* nie jest konieczna w celu odseparowania przewodu *P38*.



Rys. 5.4 Wykres przepływów w przewodzie P38 w przeprowadzonych symulacjach [opracowanie własne]

W sytuacji, gdy algorytm wyłoniłby więcej zbiorów zasuw, które powodują izolację uszkodzonego przewodu, algorytm dodaje kolejny scenariusz testowy do pliku. Na podstawie zbiorów spełniających warunek tworzony jest kolejny etap "Scenario stage (2)", w którym algorytm przeprowadza nowe symulacje hydrauliczne. Symulacje systemu zbiorowego zaopatrzenia w wodę analizują przepływ w uszkodzonym przewodzie zakładając brak dwóch zasuw (względem listy początkowej). Na podstawie nowych wyników algorytm kończy pracę wskazując możliwe rozwiązanie lub kontynuując analizę poprzez wykonanie analogicznych czynności opisanych powyżej.

W *Załączniku A* niniejszej rozprawy zawarto kod programu analizującego przepływy oraz szczegółowe wyniki symulacji dla analizowanego przykładu.

6. Podejmowanie decyzji o kolejności usuwania awarii w systemach wodociągowych

6.1 Wprowadzenie

System zbiorowego zaopatrzenia w wodę należy do grupy obiektów strategicznych całej infrastruktury miejskiej. W związku z tym, wymaga się od niego wysokiej niezawodności i nieprzerwanej ciągłości procesu dystrybucji wody zdatnej do spożycia. Zarządzanie tego typu systemami jest trudne, ponieważ wymaga spełnia dwóch przeciwstawnych celów tj. utrzymania możliwie niskich kosztów eksploatacji przy zachowaniu wysokiej niezawodności. W przypadku detekcji awarii przewodu wodociągowego, należy więc usunąć ją w jak najkrótszym możliwym czasie. Należy pamiętać, że z sytuacją wystąpienia awarii wiązać się może wiele ryzyk takich [36]: brak dostępu do wody przez dużą grupę odbiorców, znaczny wypływ wody z sieci w newralgicznych punktach, pogorszenie jakości wody w sieci, podtopienia i uszkodzenia obiektów infrastruktury miejskiej, skażenie wtórne, powstanie nowych awarii czy straty finansowe.

W przypadku wystąpienia wielu awarii istotny staje się więc proces ich klasyfikacji. Na podstawie informacji otrzymanych od osoby odpowiedzialnej za nadzór ekip naprawczych oraz dostępnej literatury, w rozdziale tym zostaną określone podstawowe informacje potrzebne do klasyfikacji awarii. Na podstawie wybranych cech awarii, zostaną określone współczynniki jej przynależności do zaproponowanych kategorii. Przynależność do konkretnej grupy będzie determinować jej ważność w procesie decyzyjnym szeregowania kolejności usuwania awarii. Dodatkowo każda z awarii będzie opisana współczynnikiem priorytetyzacji, którego wartość określać będzie ważność poszczególnej awarii.

W Rozdziale szóstym przedstawiono informację na temat sposobów klasyfikacji i priorytetyzacji awarii występujących w systemach zbiorowego zaopatrzenia w wodę. Ponadto w rozdziale w oparciu o przegląd literatury przedstawiono podstawową wiedzę z zakresu teorii szeregowania zadań, metodykę rozwiązywania problemów szeregowania zadań oraz przykłady wykorzystania algorytmów szeregowania zadań w inżynierii środowiska.

6.2 Klasyfikacja awarii

Poprzez klasyfikację rozumie się przypisanie pewnych obiektów, zjawisk (awarii) czy osób do odpowiednich klas na podstawie ich cech. Dokonując segregacji poszczególnych obiektów opisanych przez zmienną ilościową lub jakościową, konieczne jest określenie pewnych wartości tej zmiennej będących wartościami granicznymi poszczególnych klas (tzw. utworzenie schematu klasyfikacji) [121]. Podstawowym schematem klasyfikacyjnym jest tzw. *dychotomia*, w której występuje prosty podział obiektów na dwie klasy. W pierwszej klasie znajdują się obiekty posiadające pewną cechę, natomiast w drugiej klasie obiekty tej cechy pozbawione. Należy pamiętać również, że opracowany schemat klasyfikacyjny musi spełniać dwie podstawowe zasady tj. być rozłączny oraz wyczerpujący. W przypadku rozłączności, żaden z obiektów czy zjawisk nie możne znajdować się w dwóch klasach jednocześnie. Poprzez wyczerpujący natomiast rozumie się taki schemat klasyfikacyjny, który zakłada przynależność każdego obiektu do jednej z klas (brak obiektów nieprzynależących do żadnej z klas). Ze względu na wybrane kryteria klasyfikacyjne wyróżnić można klasyfikację naturalną, nienaturalną oraz sztuczną. Klasyfikacja naturalna odnosi się do nie powierzchniowych cech tego, co jest przedmiotem klasyfikacji, natomiast klasyfikacja nienaturalna jest jej przeciwieństwem. W przypadku podziału sztucznego, klasyfikacja powstaje na skutek utworzenia kategorii zaprzeczającej (negacji) klasyfikacji już istniejącej. Umiejętna klasyfikacja rzeczywistych awarii przewodów wodociągowych w sytuacji kryzysowej tj. w momencie wystąpienia awarii, których liczba przekracza liczbę dostępnych ekip naprawczych jest szczególnie ważna. Na podstawie przeglądu literaturowego oraz wiedzy osoby odpowiedzialnej za nadzór planów remontowych sieci wodociągowych oraz zadań przydzielanych poszczególnym ekipom naprawczym (dalej zwanego ekspertem) zdecydowano się na następujące klasy awarii:

- (C_1) – mająca istotny wpływ na infrastrukturę sieciową oraz proces dystrybucji wody;
- (C_2) – mająca umiarkowany wpływ na infrastrukturę sieciową oraz funkcjonowanie systemu zbiorowego zaopatrzenia w wodę;
- (C_3) – mająca niewielki wpływ na infrastrukturę wodociągową oraz proces dystrybucji wody.

Powyższe kategorie pozwolą na wstępną klasyfikację awarii z punktu widzenia przedsiębiorstwa wodno-kanalizacyjnego, którego głównym celem jest utrzymanie ciągłości procesu dystrybucji wody. Celem procesu klasyfikacyjnego jest określenie wpływu poszczególnych awarii na całą infrastrukturę wodociągową. Oznacza to, że nie są brane pod uwagę informację o punktach krytycznych (w tym priorytetyzacja).

Na podstawie informacji uzyskanych od eksperta, bazującego na wieloletnim doświadczeniu, głównym czynnikiem klasyfikacji awarii jest jej przewidywany czas naprawy (T_{PN}), który uzależniony jest od wielu zmiennych. Czas (T_{PN}), jako główny wskaźnik niezawodności, stanowiący podstawę analizy ryzyka określony został równaniem (6.1):

$$T_{PN} = T_{PP} + T_{RN} + T_{ZP} + T_{OnN} \tag{6.1}$$

gdzie:

- T_{PP} – czas potrzebny na przygotowanie podłoża [min];
- T_{RN} – czas rzeczywistej naprawy (w tym izolacja przewodu) [min];
- T_{ZP} – czas potrzebny na zabezpieczenie podłoża (czynności wykończeniowo-porządkowe) [min];
- T_{OnN} – czas oczekiwania na naprawę (czas od momentu zgłoszenia awarii do podjęcia prac naprawczych) [min].

Czas potrzebny na przygotowanie podłoża (T_{PP}) uzależniony jest przede wszystkim od czasu zabezpieczenia miejsca awarii, czasu wykonania wykopu, ewentualnego odpompowania wody oraz innych czynników utrudniających dotarcie do awarii (np. obecność pojazdów mechanicznych, drzew, słupów elektrycznych itp.). Wykonanie wykopu uzależnione jest natomiast od ukształtowania terenu, rodzaju nawierzchni i gruntu, infrastruktury podziemnej, warunków atmosferycznych oraz głębokości uszkodzonego przewodu [36].

Na czas rzeczywistej naprawy (T_{RN}) wpływają takie elementy jak: średnica oraz funkcja przewodu, rodzaj uszkodzenia, miejsce występowania awarii w strukturze sieci

wodociągowej, dostępność elementów zastępczych oraz specjalistycznego sprzętu. Ponadto do czasu rzeczywistej naprawy wlicza się czyszczenie i płukanie przewodu, odpowietrzenie, próbę szczelności, dezynfekcję oraz czas potrzebny na przywrócenie zdolności dostawczej przewodu [10, 36]. Warto nadmienić, że w przypadku wielu awarii problemy w postaci poboru wody lub znacznej utraty jej jakości nie zawsze występują w momencie ich pojawienia się. Problemy te powstają dopiero w wyniku izolacji przez ekipę naprawczą uszkodzonego przewodu, dlatego często jako brak czasu dostępu do wody uznaje się czas (T_{RN}) [36]. Czas potrzebny na zabezpieczenie podłoża (T_{ZP}) zależy od miejsca wystąpienia awarii, ukształtowania tereny, zasypania wykopu oraz innych prac porządkowych.

Tab. 6.1 *Średnie czasy trwania przykładowych czynności naprawczych [10]*

Średnie czasy trwania czynności naprawczych				
Identyfikator [ID]	**Nazwa czynności**	**Wymagany czas realizacji [min]**		
		Średnica [mm] i oznaczenie		
		< 100	100 – 300	> 300
		D_3	**D_2**	**D_1**
		przyłącze	rozdzielczy	magistrala
1	Otwarcie / Zamknięcie zasuwy	10 – 15	15 – 20	20 – 30
2	Lokalizacja awarii	20 – 25	55 – 60	50 – 60
3	Zbijanie nawierzchni asfaltowej	70 – 120	120 – 160	160 – 240
4	Ściąganie płyt chodnikowych	30 – 35	45 – 50	60 – 65
5	Wykop do rurociągu (mechaniczny)	80 – 100	100 – 140	150 – 240
6	Odpompowanie wody z wykopu	30 – 60	80 – 120	120 – 180
7	Czyszczenie przewodu	30 – 35	40 – 45	50 - 60
8	Montaż opaski naprawczej	25 – 30	40 – 50	60 – 90
9	Montaż doszczelnienia	40 – 45	50 – 55	60 – 90
10	Wycięcie odcinka przewodu	30 – 40	50 – 60	90 – 140
11	Montaż nowego przewodu	50 – 60	60 – 70	180 – 240
12	Otwarcie wody, odpowietrzenie, płukanie	20 – 25	30 – 35	50 – 60
13	Zasypanie wykopu	40 – 50	55 – 65	70 – 80
14	Czynności wykończeniowo-porządkowe	60 – 80	75 – 100	100 – 160

Przykładowy czas trwania poszczególnych czynności naprawczych przedstawiony został w Tabeli 6.1. Średnie czasy wykonywania poszczególnych zadań przygotowawczo -naprawczych opracowane zostały na podstawie ankiet przeprowadzonych przez autorów prac oraz szczegółowych danych eksploatacyjnych [10, 36]. Ankietę przeprowadzono

wśród osób odpowiedzialnych za wykonywanie poszczególnych zadań. Na podstawie powyższych danych można stwierdzić, że średni prawdopodobny czas naprawy (T_{PN}) przewodów, wynosi kolejno: (D_1) – 22 godziny i 7 minut, (D_2) – 15 godzin i 22 minuty oraz (D_3) – 10 godzin i 13 minut. Należy pamiętać, że dokładne określenie rzeczywistego całkowitego czasu naprawy jest niemożliwe ze względu na dużą liczbę zakłóceń zewnętrznych oraz wewnętrznych. Na podstawie wyliczonego czasu (T_{PN}) można stwierdzić, że wraz ze wzrostem czasu potrzebnego na naprawę wzrasta również znaczenie analizowanego uszkodzenia. Oznacza to, że awarie tego typu mają większy wpływ na całą infrastrukturę sieciową oraz proces dystrybucji wody, niż awaria, której przewidywany czas naprawy jest krótszy. W oparciu o dodatkowe dane literaturowe można również wskazać zależność między czasem naprawy awarii, a prawdopodobieństwem wystąpienia skażenia wtórnego lub wystąpienia nowych awarii [47].

Kolejnym aspektem ujętym w trakcie klasyfikacji awarii jest procentowy udział przewodu w transporcie wody w ciągu doby (P_{DPW}). Na podstawie danych o przepływach określa się krytyczność analizowanego odcinka.

Inną cechą biorącą udział w kategoryzacji awarii jest współczynnik intensywności uszkodzeń (λ), który został szczegółowo opisany w Rozdziale 3.2.4.

Następnym elementem rozpatrywanym w przypadku kategoryzacji awarii jest analiza wyników z przebiegu symulacji modelu hydraulicznego. Na podstawie informacji o czasie działania sieci z wykluczonym elementem i wiedzy na temat wygenerowanych strat ciśnienia i niedoboru wody określony zostaje współczynnik krytyczności przewodu (F_{KP}), który został szczegółowo omówiony w Rozdziale 3.2.4.

Ostatnim elementem wykorzystanym w procesie kategoryzacji awarii jest tzw. indeks zdolności systemu wodociągowego opracowany przez Todiniego [38, 39, 40]. Biblioteka WNTR umożliwia określenie zdolności systemu do przezwyciężania awarii przy jednoczesnym spełnieniu wymagań poboru i ciśnienia. Indeks Todiniego definiuje *odporność w określonym czasie jako miarę nadwyżki dostępnej wody w każdym węźle do wymaganego w nim zapotrzebowania* [39, 40].

Podsumowując, głównym wskaźnikiem determinującym klasę awarii jest jej prawdopodobny czas naprawy (T_{PN}), na który składają się elementy takie jak: czas potrzebny na lokalizację awarii, czas przygotowania podłoża, czas rzeczywistej naprawy oraz czas potrzebny na zabezpieczenie podłoża. W przypadku przygotowania podłoża na jego czas wpływa średnica przewodu, miejsce występowania oraz czas potrzebny na jego izolację. W przypadku długiego oczekiwania na naprawę należy doliczyć czas (T_{OnN}) niezbędny do odpompowania wody. Ponadto na podstawie wyżej wymienionych cech, każda awaria powinna zostać dodatkowo opisana zbiorem następujących informacji: procentowy udział przewodu w transporcie wody (P_{DPW}), wskaźnik krytyczności przewodu (F_{KP}), wskaźnik intensywności uszkodzeń (λ) oraz indeks Todiniego (I_T).

6.3 Priorytetyzacja

Szereg czynności mających wskazać, które zadania mają kluczowe znaczenie, a które są jedynie poboczne, nosi miano priorytetyzacji. Na jej podstawie można określić, które zadanie należy wykonać w pierwszej kolejności, a które odłożyć na później. W literaturze opisano wiele metod pomagających określić współczynnik priorytetyzacji poszczególnych zadań. Najpopularniejszą tego typu metodą jest tzw. metoda ABCDE, która pozwala podzielić zadania na pięć grup, które opisują konsekwencje w sytuacji nie wykonania ich w określonym czasie. Innym przykładem metody priorytetyzacji jest metoda Eisenhowera, która została opracowana na podstawie jednego z cytatów prezydenta USA brzmiącego *„To co ważne, rzadko bywa pilne, a to, co pilne, rzadko bywa ważne”*. Metoda ta pozwala przydzielić zadania do jednej z czterech grup: ważne i pilne, ważne i niepilne, nieważne i pilne oraz nieważne i niepilne.

W przypadku sytuacji kryzysowej, w której występuje wiele awarii, zwykła klasyfikacja na klasy określające ich wpływ na funkcjonowanie systemu zbiorowego zaopatrzenia w wodę nie wystarcza. Spowodowane jest to faktem określenia wyłącznie możliwych szkód systemu dystrybucji wody, a nie funkcjonowania obszaru, na którym się znajduje. To ważne, aby w tego typu sytuacjach wziąć pod uwagę infrastrukturę krytyczną (np. szpitale), które muszą mieć zapewniony stały dostęp do wody czy energii. Oznacza to, że w trakcie procesu decyzyjnego i określaniu kolejności usuwania awarii należy nadać priorytet zadaniom, które nie tylko wpływają negatywnie na SZZwW, ale również uniemożliwiają dostawę wody do punktów krytycznych.

6.4 Algorytmy szeregowania zadań w inżynierii środowiska

W celu zdefiniowania problemu szeregowania zadań należy przytoczyć dwa podstawowe pojęcia z nim związane tj. *zadanie* oraz *zasób*. Poprzez *zadanie* rozumie się realizację ciągu czynności zwanych *operacjami*, z których każda z nich wymaga wykorzystania określonych zasobów. *Zasobem* natomiast nazywa się: ludzi, maszyny, materiały itp., które są niezbędne do wykonania zdefiniowanego zadania. W obu przypadkach należy wskazać typowe cechy charakterystyczne. W sytuacji gdy mowa o zasobach, pod uwagę bierze się, czy są one odnawialne, nieodnawialne czy może są podwójnie ograniczone. Często wyróżnia się również ich dostępność, liczbę, przywłaszczalność oraz możliwość podzielności. W przypadku zadań należy natomiast określić ich termin gotowości, czas na realizację, możliwość wstrzymania zadania oraz sposób jego wykonania [83, 99].

Zgodnie z powyższym, poprzez zadanie w przypadku analizowanej tematyki rozumie się realizację naprawy awarii występującej w SZZwW, która w zależności od rodzaju uszkodzenia oraz miejsca występowania składa się z ciągu różnych czynności. Operacje w zadaniu są niepodzielne i wykonywane w określonej kolejności (brak możliwości ich zmiany). Każde zadanie można przerwać i wznowić w późniejszym terminie. Gotowość zadania determinuje czas jego zgłoszenia, natomiast żądany termin zakończenia zostaje wyznaczony na podstawie jego klasyfikacji oraz priorytetyzacji. Zasobami natomiast będą ekipy naprawcze wyposażone w specjalistyczny sprzęt niezbędny do wykonania zadania oraz materiały, które muszą zostać użyte w celu jego realizacji.

Należy pamiętać, że problem szeregowania zadań jest problemem optymalizacyjnym. Trudności w rozwiązywaniu praktycznych dyskretnych problemów optymalizacyjnych mogą wynikać z wielu czynników. Spowodowane są one obecnością dużej liczby ekstremów lokalnych, braku własności ciągłości, różniczkowalności funkcji kryterialnej oraz przynależności do klasy problemów silnie NP-trudnych. Obecnie jednak zauważyć można wzrost popularności metod przybliżonych, algorytmów ewolucyjnych czy programowania równoległego [122].

6.4.1 Metodyka rozwiązywania problemów szeregowania

W literaturze sposoby rozwiązywania problemów szeregowania zadań dzieli się na dokładne oraz przybliżone. Wynikiem działania algorytmów dokładnych są rozwiązania optymalne natomiast w przypadku algorytmów przybliżonych, rozwiązania nie zawsze są optymalne lecz często do nich zbliżone. W metodach dokładnych wyszczególnić możemy [83, 99, 122]:

- przegląd zupełny (ang. *bruteforce*) – nieefektywna metoda obliczeniowa (złożoność $\theta(2^n)$), która znajduje rozwiązanie optymalne;
- metoda podziału i ograniczeń (ang. *branch and bound*) – polegająca na badaniu (pośrednim lub bezpośrednim) wszystkich rozwiązań problemu. Jej działanie opiera się na analizie drzewa (reprezentacja wszystkich możliwych ścieżek) przestrzeni stanów. Analiza całego drzewa jest złożona obliczeniowo (złożoność wykładnicza), dlatego metoda ta oblicza granicę dla każdego węzła, która pozwala na jego ocenę;
- programowanie całkowitoliczbowe (ang. *programming linear constraints*) – metoda programowania liczbowego, w której narzucono dodatkowe ograniczenia na pewne zmienne decyzyjne. Cel tego typu ograniczeń uwarunkowany jest stanem rzeczywistym (np. nie istnieje ½ osoby). Problem programowania liniowego całkowitoliczbowego jest NP-trudny;
- programowanie dynamiczne (ang. *dynamic programming*) – w skrócie polegające na przekształceniu problemu optymalizacyjnego na wieloetapowy proces podejmowania decyzji, w którym stan każdego etapu uzależniony jest od podjętej decyzji (wybranej ze zbioru decyzji dopuszczalnych).

W przypadku metod przybliżonych, wykorzystywanych w sytuacjach, w której istotną rolę odgrywa czas wyróżniamy przede wszystkim takie algorytmy jak [99]:

- algorytmy przeszukiwania lokalnego – zadaniem, których jest analiza rozwiązań sąsiednich, na podstawie których wyznaczane są rozwiązania kolejne. W sytuacji spełnienia określonych warunków stopu algorytmu

przez wyznaczone rozwiązanie, następuje przerwanie procesu przeszukiwania. Elementarnym kryterium stopu w tego typu rozwiązaniach jest osiągnięcie minimum lokalnego;

- algorytmy konstrukcyjne – opierające się na tzw. metodzie wstawień, która składa się z dwóch etapów. Etap pierwszy (wstępny) określa listę początkowych czynności, które należy wykonać (lista określana jest na podstawie algorytmu wykorzystującego reguły priorytetowe). Etap drugi natomiast odpowiada za wygenerowania ciągu *n* permutacji częściowych w oparciu o listę początkowych czynności. Na podstawie zbioru rozwiązań próbnych wybierane jest to, które otrzymało najlepszą ocenę wartości funkcji celu [37];
- schematy aproksymacyjne – wynikiem których jest zbiór rozwiązań o zadeklarowanej dokładności;
- algorytmy metaheurystyczne – wykorzystujące elementy sztucznej inteligencji. Działanie tego typu algorytmów zbliżone jest do działania algorytmów przeszukiwania lokalnego, jednakże posiadają dużo bardziej złożone metody przeglądania rozwiązań sąsiednich. Dodatkową zaletą tego typu algorytmów jest umiejętność opuszczenia minimum lokalnego.

Należy pamiętać, że w przypadku algorytmów przybliżonych konieczna jest ich ewaluacja z punktu widzenia jakości otrzymywanych wyników. W tego typu zadaniach wykorzystuje się analizę probabilistyczną oraz analizę najgorszego przypadku. W sytuacji analizy probabilistycznej wskazuje się statystyczną zbieżność algorytmu w stosunku do rozwiązania optymalnego, natomiast w drugim przypadku wyznacza się różnicę między rozwiązaniem najlepszym oraz najgorszym [99].

Podsumowując, w celu znalezienia rozwiązania problemu szeregowania zadań należy go przede wszystkim zdefiniować w sposób szczegółowy oraz zaimplementować algorytm rozwiązania, który będzie dla niego odpowiedni (efektywny obliczeniowo). Dla problemów o złożoności wielomianowej powinno stosować się optymalne algorytmy wielomianowe, natomiast dla problemów klasy silnie NP-trudnych rozważyć zastosowanie algorytmów przybliżonych lub pseudowielomianowych [83, 99].

6.4.2 Wykorzystanie algorytmów szeregowania zadań w inżynierii środowiska

Algorytmy szeregowania zadań wykorzystywane są w wielu dziedzinach życia. Z roku na rok ze względu na występowanie zjawiska konkurencyjności problematyka ta cieszy się coraz większym zainteresowaniem ze strony nauki, techniki oraz prywatnych przedsiębiorstw. To właśnie dzięki możliwościom zamodelowania pewnych procesów zachodzących w danym przedsiębiorstwie można rozwiązać szereg problemów, co wpływa na efektywność jego działania. Rozdział ten poświęcony został przedstawieniu przykładów wykorzystania algorytmów szeregowania zadań w inżynierii środowiska, a dokładniej w modelowaniu procesów wodno-ściekowych.

Pierwszym przykładem wykorzystania problematyki szeregowania zadań jest organizacja pracy laboratorium przy inwentaryzacji zrzutu ścieków z zakładów przemysłowych, który został szerzej opisany w pracy [99]. W ww. artykule szczegółowo został przedstawiony proces zbierania i analizy próbek ścieków przez wykwalifikowane osoby. Na podstawie znajomości procesu zdefiniowano maszyny oraz operacje. Następnie na podstawie pożądanego przedziału zakończenia wykonywania zadań opracowany został harmonogram, którego celem była minimalizacja kosztów ponoszonych na wypożyczenie zaawansowanej aparatury badawczej niezbędnej do badania zebranych próbek.

Kolejnymi przykładami są prace [16, 107]. W pierwszej z nich głównym celem było wskazanie, że optymalizacje z wykorzystaniem algorytmu baysesowskiego można traktować jako ogólne ramy dla modelowania opartego na danych i rozwiązywania problemów pojawiających się w systemach zbiorowego zaopatrzenia w wodę. Argumentem potwierdzającym powyższe stwierdzenie była implementacja algorytmu rozwiązującego problem ponoszonych kosztów i strat energii w planowaniu rozmieszczenia i harmonogramowaniu pracy pomp. Druga praca porusza podobną tematykę (optymalizacja harmonogramowania pracy pomp w sieciach wodociągowych), jednakże w celu znalezienia rozwiązania wykorzystano algorytm głębokiego uczenia się ze wzmocnieniem. Powyższy problem został zamodelowany jako proces decyzyjny Markova, biorący za cel zarządzanie ciśnieniem. Skuteczność i stosowalność zaproponowanego rozwiązania przedstawiono na przykładzie 22 węzłowej sieci z dwiema pompami.

Innym przykładem opisującym wykorzystanie problematyki szeregowania zadań jest praca [6], w której przedstawiono problemy związane z przerwami procesu dystrybucji wody. Opisano również wykorzystanie narzędzi EPANET i WNTR do symulacji ww. procesu oraz sytuacji krytycznej (trzęsienia ziemi). Na podstawie modelu hydraulicznego sieci wodociągowej, informacji wstępnych takich jak: lista awarii, liczba dostępnych ekip remontowych, dostępność urządzeń, opracowany algorytm określał kolejność napraw do wykonania. Analizowaną funkcją celu w powyższym artykule była minimalizacja czasu potrzebnego na przywrócenie ciągłości procesu dystrybucji wody.

W artykule [11] przedstawiono metodę definiowania rankingu przewodów wodociągowych, w celu uzyskania najlepszych współczynników dla zdefiniowanych funkcji celu (kryteriów), dzięki czemu możliwe będzie dostarczenie rozsądnego harmonogramu (kolejności) sekwencjonowania napraw. Zaproponowana metoda tworzenia rankingu przewodów i ich oceny oparta została na wielokryterialnej metodzie decyzyjnej (metody organizacji rankingu preferencji do oceny wzbogacenia – PROMETHEE). Metoda została przeanalizowana na pięciu różnych scenariuszach katastrof, które zostały dostarczone przez komitet organizacyjny „*The Battle of Post-Disaster Response and Restoration*".

Ostatnim przykładem jest artykuł [32], w którym przedstawiono metodologię optymalnego projektowania i harmonogramowania inwestycji związanych z naprawą sieci wodociągowych. Metoda została opracowana na podstawie ewolucyjnej techniki programowania SMGA (ang. *Structured Messy Genetic Algirthms*), która wykorzystuje wielocelową formułę usprawniającą proces ewolucyjny. Rozwiązanie zostało przetestowane i zweryfikowane na sztucznie zaprojektowanej sieci składającej się 15 przewodów.

7. Efektywne algorytmy przywracania ciągłości dostawy wody

7.1 Wprowadzenie

Mimo dynamicznego rozwoju techniczno-technologicznego, wiedzy i świadomości o procesach zachodzących w środowisku trudno jest przewidzieć kiedy, gdzie i w jakim stopniu może wystąpić sytuacja kryzysowa. Bez względu na znajomość przyczyn powstawania katastrof nikt nie jest w stanie ich wyeliminować. Istnieje jedynie możliwość ograniczenia jej skutków. W przypadku infrastruktury krytycznej, do której należy system zbiorowego zaopatrzenia w wodę, modelowanie pracy tego typu obiektów w sytuacjach kryzysowych wymaga stosowania złożonych algorytmów szeregowania zadań dla dostępnych ekip naprawczych.

W rozdziale przedstawiono grupę algorytmów stanowiących szkielet systemu wspomagania decyzji, których zadaniem jest poprawa efektywności procesu przywracania ciągłości dostawy wody w sytuacji po katastroficznej. Punktem wyjściowym do opracowania metodyki była analiza doświadczeń eksperta dziedzinowego oraz zbiór artykułów naukowych.

7.2 Algorytm klasyfikacji elementów sieci wodociągowej

Podstawową informacją niezbędną w procesie klasyfikacji awarii i określenia ich stopnia ważności jest wiedza o systemie wodociągowym, w którym owe awarie wystąpiły. W celu określenia krytyczności elementów wchodzących w skład sieci zaproponowano algorytm ich klasyfikacji (dalej zwany *Algorytmem 1*). Zadaniem zaimplementowanego algorytmu jest testowanie za pośrednictwem symulacji sposobu funkcjonowania sieci wodociągowej w przypadku braku analizowanych elementów.

Pierwszym krokiem algorytmu 1 jest utworzenie obiektu przechowującego model hydrauliczny analizowanej sieci wodociągowej. W tym celu należy odpowiednio uzupełnić plik konfiguracyjny *(config.py*), który przechowuje informację takie jakie:

- nazwa analizowanego modelu hydraulicznego;
- czas trwania symulacji;

- ścieżki dostępu do:
    - modelu hydraulicznego;
    - dodatkowych informacji o modelu;
    - plików wynikowych;
    - rozmieszczenia zasuw w sieci;
    - scenariuszy testowych i opisu awarii;
- kategorie klasyfikacji oraz ich wartości progowe;
- czynności naprawcze i czas ich trwania.

Plik konfiguracyjny (*config.py*) dla omawianego w Rozdziale 6 i 7 modelu *ariel_phd.inp* został zamieszczony w *Załączniku B* niniejszej rozprawy.

Kolejnym krokiem algorytmu jest uruchomienie symulacji referencyjnej (bazowej), której zadaniem jest przetestowanie funkcjonowania sieci wodociągowej w warunkach normalnych (bez awarii). Wyniki symulacji bazowej zostają zapisane do wskazanych w pliku konfiguracyjnym plików o rozszerzeniu .CSV, które przechowują między innymi informacje o przepływach w poszczególnych przewodach sieci, dobowym przepływie w każdym z przewodów, ciśnieniu w każdym węźle, dobowym ciśnieniu w poszczególnych węzłach, poborze wody w węzłach oraz raport symulacyjny wygenerowany przez silnik obliczeń. Po zakończeniu symulacji bazowej wygenerowana zostaje lista elementów wchodzących w skład analizowanego modelu hydraulicznego. Powyższa lista zawiera informacje o liczbie i identyfikatorach: rezerwuarów, zbiorników, pomp, zasuw oraz przewodów. Utworzenie obiektu klasy Algorytmu 1 oraz wywołanie metody odpowiedzialnej za uruchomienie symulacji bazowej przedstawiono poniżej:

```
# =========== Utworzenie obiektu testowego:
new_case = Algorithm_1()

# =========== Uruchomienie symulacji bazowej:
ref_sim = new_case.reference_simulation()
```

Następnym krokiem jest wywołanie metody odpowiedzialnej za przeprowadzenie symulacji braku poszczególnych elementów sieci wodociągowej. Użytkownik może

samodzielnie wskazać listę elementów, które mają zostać poddane testom, lub uruchomić ją automatycznie dla wszystkich elementów wskazanego typu. Poniżej przedstawiono przykład wywołania metody analizy braku rezerwuaru w obu konfiguracjach:

```
# =========== Testowanie sieci dla wskazanych rezerwuarów:
res_sim = new_case.reservoir_resilience_simulation(['R1'])

# =========== Testowanie sieci dla wszystkich rezerwuarów:
res_sim = new_case.reservoir_resilience_simulation()
```

Pierwszy przykład spowoduje wykonanie symulacji braku w sieci rezerwuaru o identyfikatorze R1, natomiast drugi przykład wykona symulację dla wszystkich rezerwuarów istniejących w analizowanym modelu. Oznacza to, że w konfiguracji drugiej algorytm wygeneruje listę *n* modeli hydraulicznych (gdzie *n* to liczba rezerwuarów), w którym każdy z modeli będzie pozbawiony jednego z rezerwuarów. Wszystkie wyniki symulacji zapisywane są również do pliku .CSV. Algorytm umożliwia również wywołanie następujących metod testowych:

```
# =========== Testowanie braku rezerwuarów:
res_sim = new_case.reservoir_resilience_simulation()

# =========== Testowanie braku zbiorników:
tan_sim = new_case.tank_resilience_simulation()

# =========== Testowanie braku pomp:
pom_sim = new_case.pomp_resilience_simulation()

# =========== Testowanie braku zasuw:
val_sim = new_case.valve_resilience_simulation()

# =========== Testowanie braku przewodów:
pip_sim = new_case.pipe_resilience_simulation()
```

W celu poprawnego działania symulacji konieczne jest zapewnienie poprawności topologicznej analizowanej sieci. Oznacza to, że nie można usunąć elementów sieci wodociągowej reprezentowanych jako węzły (rezerwuary i zbiorniki). Samo usunięcie tego typu elementów spowoduje błąd występowania w sieci elementu (przewodu), którego nieznany jest węzeł początkowy lub końcowy. W tym celu elementy takie jak rezerwuar czy zbiornik zastępowane są węzłami o zerowym poborze wody. Tego typu zabieg umożliwi uruchomienie symulacji hydraulicznej, ponieważ z punktu widzenia

topologicznego sieć jest skonstruowana poprawnie. Analogicznie wygląda sytuacja w momencie usunięcia zasuwy, pompy czy przewodu. W sytuacji, gdy usunięcie jednego z wyżej wymienionych elementów spowoduje błędy topologiczne to w przypadku:

- zasuw i pomp – element zamieniany jest na przewód o średnicy i długości równej długości i średnicy usuwanego elementu;
- przewodów – algorytm usuwa węzeł, który w wyniku usunięcia przewodu nie posiada podłączenia do innego przewodu (informacja tego typu zapisywana jest w wynikach symulacji).

Tab. 7.1 *Przykład wyników symulacji Algorytmu 1 [opracowanie własne]*

Przykład wyników testowych dla przewodów w sieci ariel_phd (przepływ [l/s])									
ID	**P1**	**P10**	**P11**	**…**	**P24**	**P28**	**Suma**	**%**	**Kategoria**
(Ref_sim)	**1,95**	**1315,21**	**1005,45**	**…**	**-131,68**	**290,22**	**6938,67**	**100**	CAT-1
(Res_R1)	-6E-08	-222,08	-467,50	…	848,05	-71,4	1082,56	15,61	CAT-4
(Tan_T1)	1,95	1315,21	1005,45	…	-131,68	290,22	6938,66	100	CAT-1
(Tan_T2)	1,95	1315,20	1005,42	…	-131,69	290,28	6938,80	100	CAT-1
(Pip_P1)	0	-222,08	-467,50	…	848,05	-71,46	1082,58	15,61	CAT-4
(Pip_P10)	1,83	0	490,78	…	-58,34	243,96	5100,31	73,51	CAT-3
(Pip_P11)	1,91	883,89	0	…	-85,07	194,92	4427,63	63,81	CAT-3
(Pip_P12)	1,95	1228,98	743,55	…	-90,41	109,58	4577,74	64,25	CAT-3
(Pip_P15)	1,95	1702,69	1216,43	…	-122,58	336,9	7316,92	105,45	CAT-1
…	…	…	…	…	…	…	…	…	…
(Pip_P22)	1,95	1341,31	1148,38	…	-124,93	310,2	7192,02	103,65	CAT-1
(Pip_P23)	1,95	1317,76	1014,20	…	-123,53	262,89	6879,09	99,14	CAT-1
(Pip_P18)	1,95	1289,69	926,92	…	86,78	156,53	6223,74	89,7	CAT-2
(Pip_P29)	1,95	1315,16	1005,29	…	-113,39	310,44	6951,21	100,18	CAT-1
(Pip_P40)	1,95	1315,21	1005,44	…	-131,63	290,28	6938,87	100	CAT-1

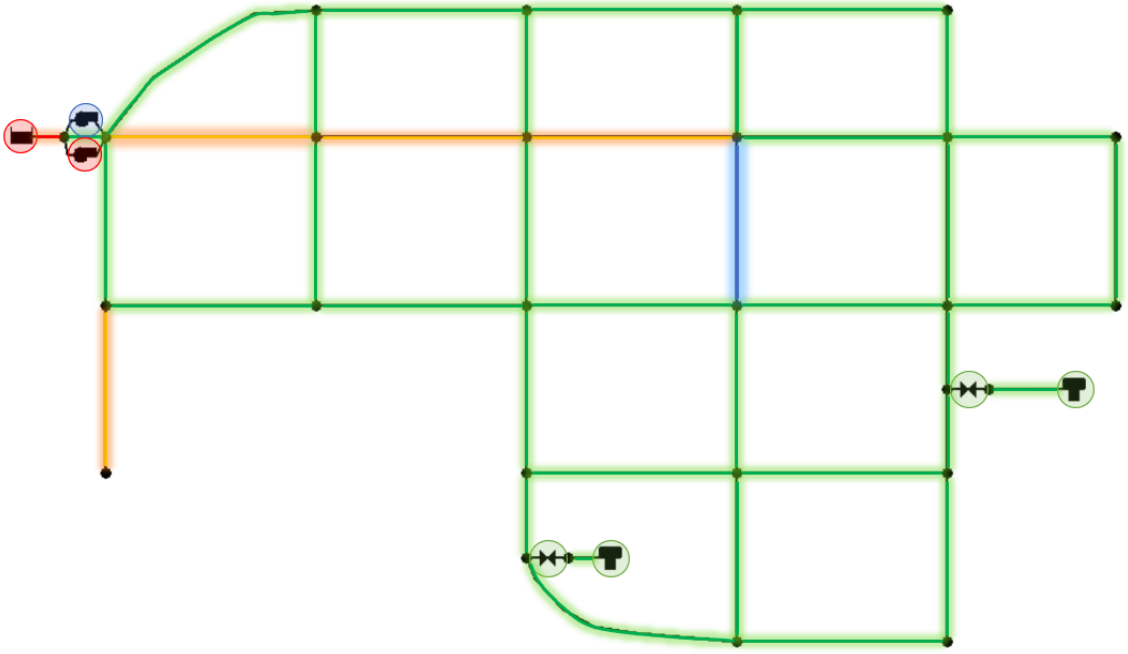
Po wykonaniu wszystkich symulacji zawartych w *scenariuszu testowym*[6] i zapisaniu wyników do plików .CSV możliwe jest ich porównanie z wynikami referencyjnymi. Fragment przykładowych wyników symulacji algorytmu klasyfikacji elementów sieci wodociągowej przedstawiono w Tabeli 7.1. Pierwsza kolumna wyników zawiera identyfikator symulacji (np. Pip_P31) oznaczający przypadek testowania sieci z wykluczeniem elementu o wskazanym identyfikatorze (np. przewód o identyfikatorze P31). W kolejnych kolumnach znajdują się dobowe przepływy w poszczególnych

[6] poprzez *scenariusz testowy* w tym przypadku rozumie się zbiór metod wywołanych w algorytmie.

przewodach, zasuwach i pompach. Trzecia kolumna od końca zawiera sumę dobowych przepływów, przedostatnia procentowy dobowy przepływ wody względem wyników wzorcowych. Ostatnia kolumna zawiera informację o przydzielonej kategorii krytyczności elementu. Należy pamiętać, że algorytm umożliwia analizę sieci pod względem przepływów, ciśnienia, zapotrzebowania oraz wysokości słupa cieczy wody w poszczególnych węzłach. Przykładowo analizując sieć pod względem wpływu elementów wchodzących w skład SZZwW na przepływ wody w sieci, dostępne są cztery kategorie:

- CAT-1: jeśli wynik dobowego przepływu wody jest w zakresie od 90% wzwyż względem wyników wzorcowych. Brak elementu nie stanowi dużego zagrożenia w funkcjonowaniu sieci;
- CAT-2: jeśli wynik dobowego przepływu wody jest w zakresie od 75% do 89% względem wyników wzorcowych – brak elementu stanowi umiarkowane zagrożenie w funkcjonowaniu sieci;
- CAT-3: jeśli wynik dobowego przepływu wody jest w zakresie od 50% do 74% względem wyników wzorcowych – brak elementu stanowi wysokie zagrożenie w funkcjonowaniu sieci;
- CAT-4: jeśli wynik dobowego przepływu wody w analizowanym przypadku jest poniżej 50% względem wyników wzorcowych – brak elementu stanowi katastroficzne zagrożenie w funkcjonowaniu sieci.

Wartości poszczególnych progów oraz liczbę kategorii można dowolnie zmieniać w pliku konfiguracyjnym (*config.py*). Wyniki działania *Algorytmu 1* dla przykładowego modelu sieci wodociągowej można również przedstawić w postaci graficznej (patrz rys. 7.1).



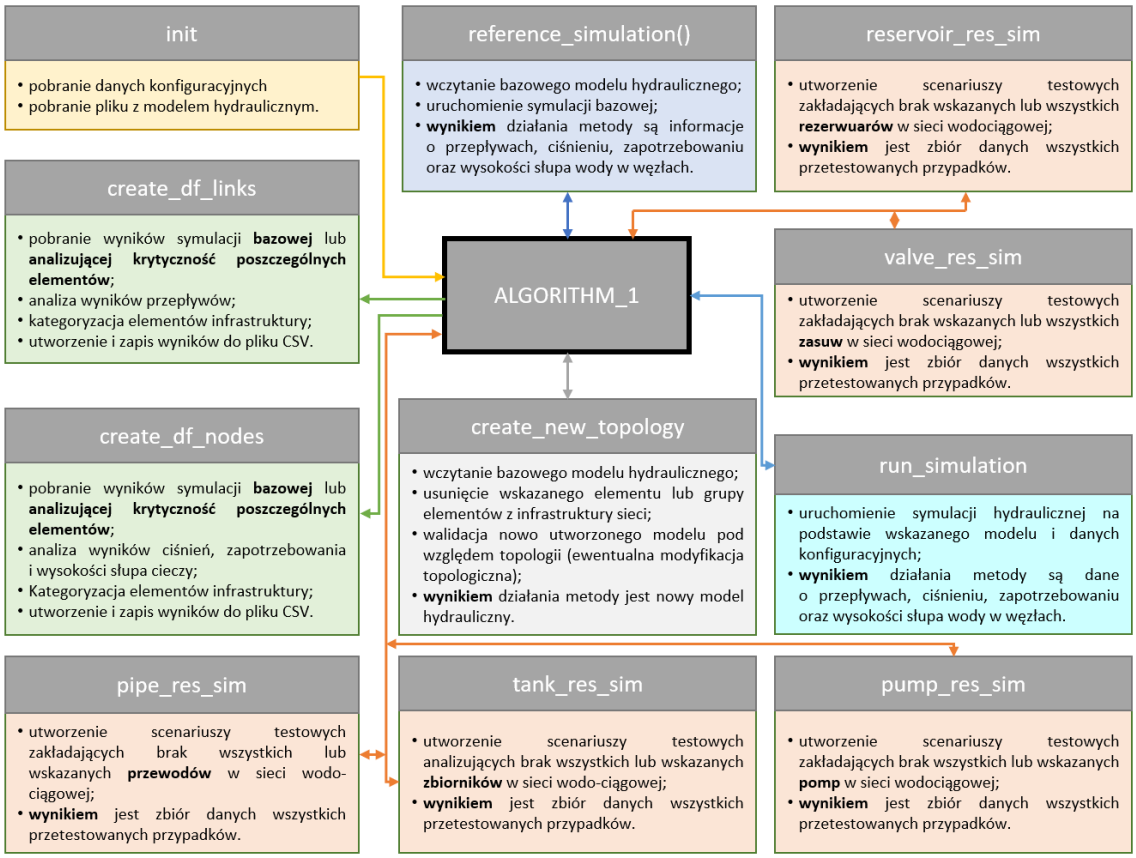
Rys. 7.1 Przykład reprezentacji graficznej działania algorytmu klasyfikacji elementów sieci wodociągowej [opracowanie własne]

Na rys. 7.1 kolorem zielonym oznaczono elementy, których brak nieznacznie wpływa na funkcjonowanie sieci (CAT-1), kolorem niebieskim (CAT-2) elementy o niewielkim wpływie na proces dystrybucji. Pomarańczowy kolor oznacza wysoką krytyczność danego elementu (CAT-3), natomiast kolor czerwony elementy bez których sieć nie może funkcjonować (CAT-4). Prezentowana sieć ma charakter szkieletowy. Oznacza to, że jest wysoce odporna na awarie. W przypadku awarii jednego przewodu inne przewody są w stanie sprostać wymaganemu zapotrzebowaniu i dostarczyć wodę do wszystkich węzłów. Algorytm nie bierze pod uwagę rozmieszczenia zasuw i konieczności izolacji pewnych sektorów sieci w sytuacjach renowacji lub naprawy. Możliwe jest wystąpienie sytuacji, w której element o niewielkim wpływie na funkcjonowanie sieci w procesie naprawczym poprzez izolację sektorów odetnie dostęp do wody dużej grupie odbiorców.

Analiza krytyczności elementów wchodzących w skład sieci wodociągowej z punktu widzenia zapotrzebowania, ciśnienia oraz wysokości słupa cieczy przebiega analogicznie jak w przypadku przepływu. Na etapie określania kategorii poszczególnych

właściwości należy pamiętać, że w przypadku ciśnienia jego znaczny wzrost może uszkodzić sieć, warto więc monitorować nie tylko jego spadek ale również wzrost.

Na rys. 7.2 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku C* niniejszej rozprawy zawarto kod źródłowy algorytmu klasyfikacji elementów systemu zbiorowego zaopatrzenia w wodę (*Algorytm 1*).



Rys. 7.2 Schemat przepływu danych *Algorytmu 1* [opracowanie własne]

Poprawność implementacji algorytmu klasyfikacji elementów sieci wodociągowej (*Algorytm 1*) zweryfikowano w oparciu o model hydrauliczny zamodelowany w Rozdziale 4.3. Testowy model hydrauliczny (*ariel_phd.inp*) składa się z 27 węzłów, 40 przewodów, 2 zbiorników, 1 rezerwuaru, 2 pomp oraz 2 zasuw. Na podstawie topologii sieci przygotowano zbiór 20 przypadków testowych, którym został poddany algorytm. Następnie w oparciu o utworzone przypadki testowe wygenerowano 20

zmodyfikowanych modeli wykorzystując oprogramowanie EPANET (sposób tworzenia i edycji modeli opisany został szerzej w Rozdziale 4.3.2). Każdy model został uruchomiony w sposób manualny, po czym wyniki porównano z wynikami algorytmu. Analiza wyników potwierdziła poprawność działania zaproponowanego rozwiązania.

7.3 Algorytm wyznaczania tras przepływu wody (cel – źródło)

Znajomość trasy przepływu wody od źródła (lub kliku źródeł) do wskazanego węzła stanowi kluczową wiedzę w procesie klasyfikacji i priorytetyzacji awarii w sytuacjach kryzysowych. W sytuacji wielu awarii niezbędny staje się proces decyzyjny, który w oparciu o tego typu algorytm może znacznie ułatwić jej podjęcie. Celem algorytmu wyznaczania tras przepływu wody (dalej zwanego *Algorytmem 2*) jest określenie trasy podstawowej wody od źródła do wskazanego celu. Algorytm wskazuje również trasy pośrednie wody do węzła (np. z innych źródeł).

Analogicznie jak w przypadku algorytmu klasyfikacji elementów sieci wodociągowej, pierwszym krokiem *Algorytmu 2* jest utworzenie obiektu przechowującego model hydrauliczny analizowanej sieci wodociągowej (dane inicjalizacyjne pobierane są bezpośrednio z pliku konfiguracyjnego *config.py*).

Drugim krokiem algorytmu po utworzeniu obiektu jest uruchomienie jednej z dwóch dostępnych metod. Pierwsza metoda (*path_creator*) na podstawie identyfikatora węzła zwraca trasę (zbiór identyfikatorów przewodów) wody z dostępnych źródeł (rezerwuarów lub zbiorników) do wskazanego węzła. Druga metoda (*critical_pipes*) natomiast informuje na podstawie trasy i wartości przepływów, o ważności przewodu w procesie dystrybucji wody do danego węzła. Sposób utworzenia obiektu i przykładowe wywołanie obu metod przedstawiono poniżej:

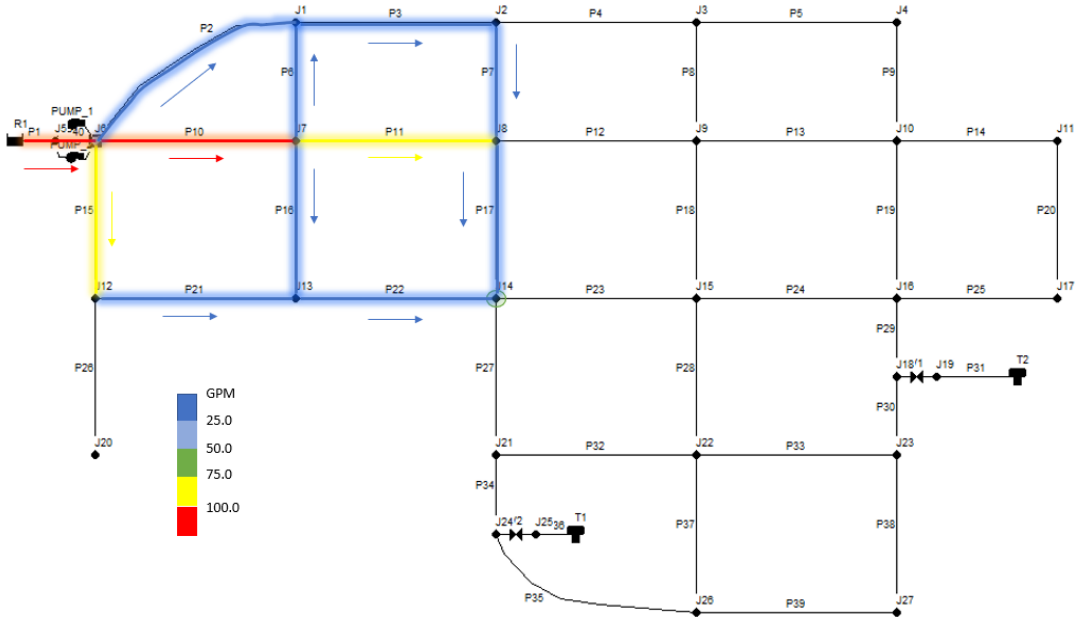
```
# ========== Utworzenie obiektu testowego:
new_case = Algorithm_2()

# ========== Wywołanie metody path_creator dla węzła J14:
new_case.path_creator(node_id='J14')

# ========== Wywołanie metody critical_pipes dla węzła J14:
new_case.critical_pipes(node_id='J14')
```

Pierwszym zadaniem metody *path_creator()* jest określenie węzłów sąsiadujących z analizowanym węzłem. Na podstawie topologii sieci wyznaczane są identyfikatory sąsiadów oraz identyfikatory przewodów łączących je z analizowanym węzłem. W przypadku węzła J14 (rys. 7.2) węzłami sąsiadującymi są węzły: J8 (przewód łączący: P17), J13 (przewód łączący: P22), J15 (przewód łączący: P23) oraz węzeł J21 (przewód łączący: P27). W pierwszej iteracji algorytmu na podstawie powyższej listy uruchamiana jest symulacja hydrauliczna, której zadaniem jest określenie prędkości przepływu w poszczególnych przewodach (określonych w powyższym etapie) w każdym kroku symulacji. Na podstawie wyników symulacji algorytm wyznacza w zbiorze sąsiadów, węzły będące „biorcami" oraz „dawcami". Węzły oznaczone jako biorcy odpowiadają za pobieranie wody od węzła analizowanego, natomiast dawcy to zbiór węzłów dostarczających wodę do analizowanego węzła. Pierwsza iteracja wyznacza powyższe zbiory (biorców i dawców) dla każdego kroku symulacyjnego. Przykładowo dla pierwszej godziny symulacji węzły o identyfikatorach: J15 oraz J21 są biorcami (pobierają wodę z węzła J14), natomiast węzły J8 i 13 należą do dawców (dostarczają wodę do węzła J14). W kolejnej iteracji w sposób równoległy analizuje się zbiór dawców (J8 i J13). Na podstawie wyników symulacji drugiej iteracji algorytmu uzyskuje się nowe zbiory informacji.

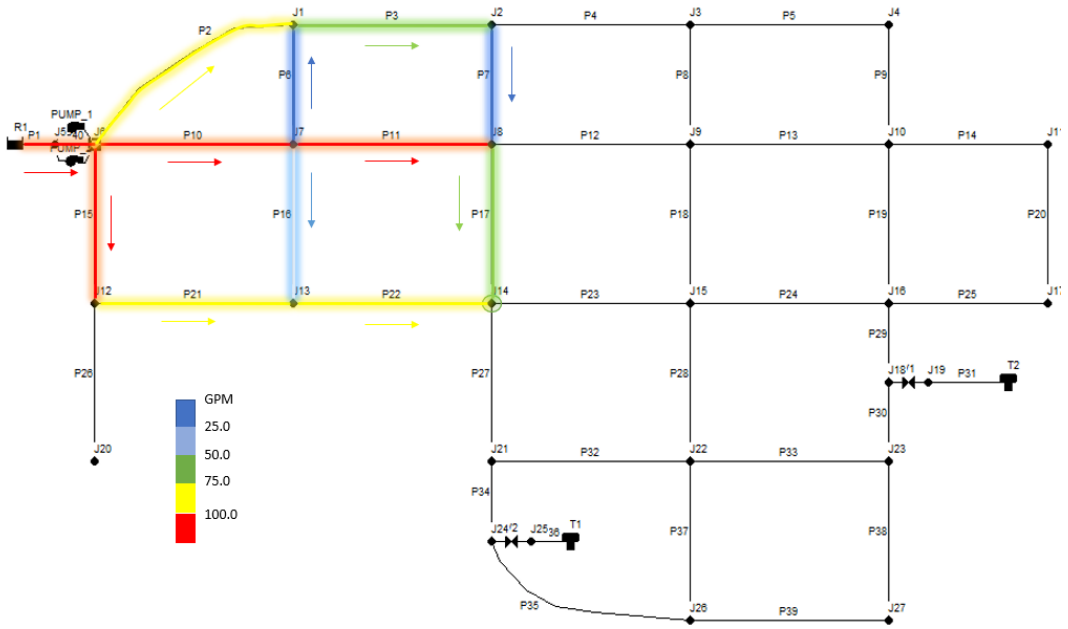
Węzeł J8 posiada 4 sąsiadów o identyfikatorach: J2, J7, J9 oraz J14. Węzeł J14 nie jest poddawany analizie w tej iteracji. Na podstawie analizy wartości przepływów w pierwszej godzinie symulacji węzły J2 i J7 zakwalifikowano do grupy biorców (węzeł J2 – przewód łączący P7 oraz węzeł J7 – przewód łączący P11). Węzeł J13 posiada 3 sąsiadów o identyfikatorach: J7, J12 oraz J14. Węzeł J14 nie jest poddawany analizie w tej iteracji. Na podstawie analizy wartości przepływów w pierwszej godzinie symulacji oba węzły zakwalifikowano do grupy biorców (węzeł J7 – przewód łączący P16 oraz węzeł J12 – przewód łączący P21). Kolejne iteracje algorytmu wykonywane są w sposób analogiczny.

W powyższy sposób metoda (*path_creator*) wyznacza trasę od wskazanego celu do źródła. Metoda (*critical_pipes*) dodatkowo w stosunku do metody pierwszej analizuje wartości przepływu na podstawie których określa stopień ważności dawców. Takie podejście pozwoli na określenie stopnia ważności poszczególnych przewodów

w procesie dystrybucji wody do wskazanego węzła. Na rys. 7.3 oraz rys. 7.4 przedstawiono wyniki symulacji (w postaci graficznej) algorytmu wyznaczania tras przepływu wody dla analizowanego przykładu w pierwszej i siódmej godzinie symulacji.



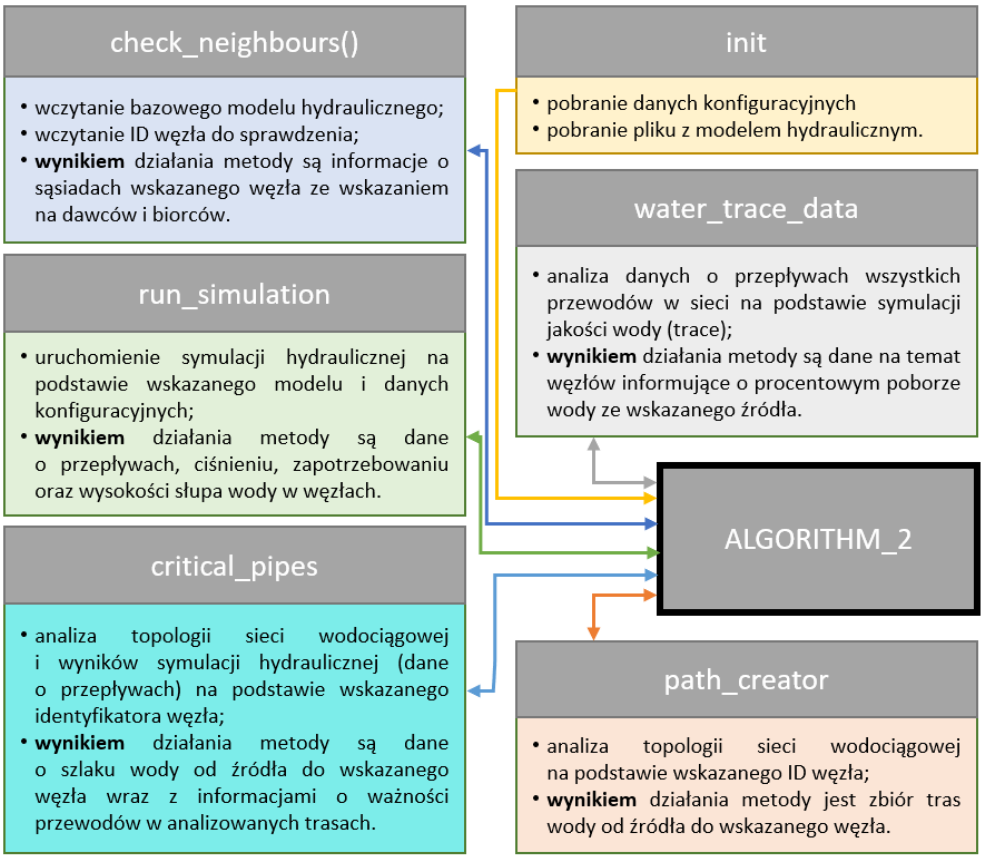
Rys. 7.3 Przykład reprezentacji graficznej działania algorytmu wyznaczania trasy dla pierwszej godziny symulacji [opracowanie własne]



Rys. 7.4 Przykład reprezentacji graficznej działania algorytmu wyznaczania trasy dla siódmej godziny symulacji [opracowanie własne]

Należy zwrócić uwagę, że stopień ważności przewodu jest zmienny i uzależniony od godziny symulacji i oczekiwanego poboru wody.

Na rys. 7.5 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku D* niniejszej rozprawy zawarto kod *Algorytmu 2*.



Rys. 7.5 Schemat przepływu danych *Algorytmu 2* [opracowanie własne]

Poprawność implementacji *Algorytmu 2* zweryfikowano w oparciu o model hydrauliczny opisany w Rozdziale 4.3 oraz podstawowy model testowy programu EPANET o nazwie *Net3.inp*. Na podstawie topologii sieci przygotowano zbiór 6 przypadków testowych (po 3 dla każdego modelu), którym został poddany algorytm. Następnie w oparciu o utworzone przypadki testowe uruchomiono i przeanalizowano wyniki symulacji systemu dystrybucji w oprogramowaniu EPANET. Każdy model został

uruchomiony w sposób manualny, po czym wyniki porównano z wynikami algorytmu. Analiza wyników potwierdziła poprawność działania zaproponowanego rozwiązania.

7.4 Algorytm typowania zasuw do zamknięcia z analizą przepływów

Metodyka działania algorytmu typowania zasuw do zamknięcia została przedstawiona w Rozdziale 5.3, w którym to na podstawie pracy doktorskiej [35] została zaimplementowana metoda macierzowo-grafowa. Powyższa metoda została uzupełniona o analizę przepływów opracowaną przez autora, co umożliwiło wyznaczenie minimalnego zbioru zasuw niezbędnych do odizolowania wskazanego odcinka. Zmodyfikowane podejście [35] nosi nazwę algorytmu typowania zasuw do zamknięcia z analizą przepływów (dalej *Algorytm 3*). Pierwszym krokiem algorytmu jest utworzenie obiektu przechowującego informację o modelu hydraulicznym analizowanej sieci. Drugim krokiem jest natomiast utworzenie trzech macierzy sąsiedztwa (A, B, C) przechowujących informacje o zależności między węzłami i przewodami oraz rozmieszczeniu zasuw na sieci. Informację o zasuwach pobierane są z przygotowanego pliku w formacie .JSON (lokalizacja pliku pobierana jest z pliku konfiguracyjnego *config.py*). Zawartość pliku z informacjami o zasuwach w analizowanym przykładzie umieszczona została w *Załączniku E*. Utworzenie nowego obiektu oraz wywołanie metod odpowiedzialnych za wygenerowanie macierzy sąsiedztwa przedstawiono poniżej:

```
# =========== Utworzenie obiektu testowego:
new_case = Algorithm_3()

# =========== Wywołanie metody tworzącej Macierz A:
new_case.matrix_a_creator()
# =========== Wywołanie metody tworzącej Macierz B:
new_case.matrix_b_creator()
# =========== Wywołanie metody tworzącej Macierz C:
new_case.matrix_c_creator()
```

Po utworzeniu macierzy istnieje możliwość wywołania czterech dostępnych metod:

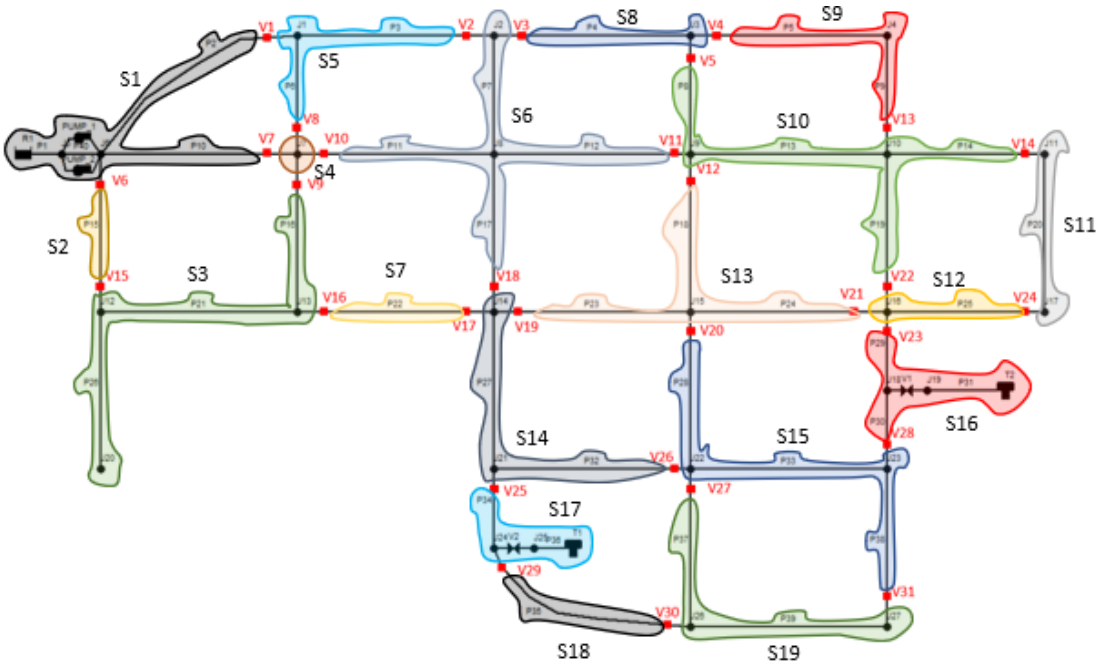
- *find_segment*, która na podstawie wskazanego identyfikatora przewodu określa zbiór elementów wchodzących w skład segmentu;
- *find_all_segments*, która analizując macierze wyznacza wszystkie segmenty w analizowanej sieci wodociągowej;

- *save_segments*, która zapisuje informację o wszystkich segmentach do pliku z rozszerzeniem .JSON;
- *min_number_of_valves*, która działa analogicznie do metody *find_segment*, ale dodatkowo na podstawie symulacji hydraulicznych wyznacza minimalny zestaw zasuw do zamknięcia.

Przykład wywołania powyższych metod wygląda następująco:

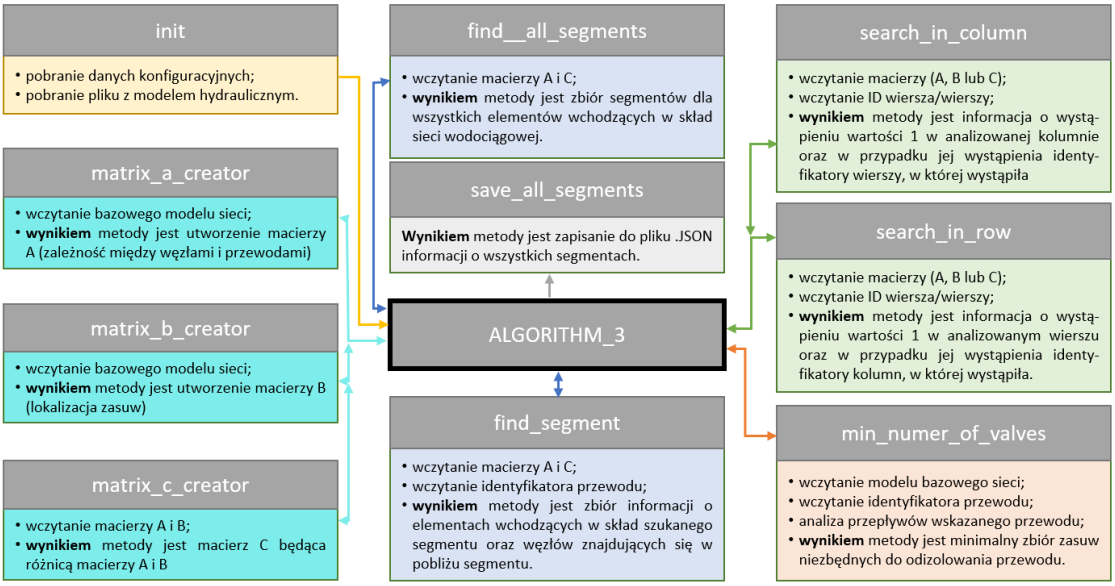
```
# =========== Wyznaczenie segmentu dla przewodu P38:
new_case.find_segment('P38')

# =========== Wywołanie metody wyznaczającej wszystkie segmenty:
new_case.find_all_segments()

# =========== Wywołanie metody zapisującej dane o segmentach:
new_case.save_segments()

# =========== Min. zbiór zasuw do odizolowania przewodu P38
new_case.min_number_of_valves('P38')
```

Po wywołaniu metody *find_all_segments()* dla przykładowego modelu hydraulicznego algorytm odnalazł 19 segmentów (patrz rys. 7.6).



Rys. 7.6 Wynik działania algorytmu wyznaczającego segmenty w analizowanym systemie dystrybucji wody [opracowanie własne]

Na rys. 7.7 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku F* niniejszej rozprawy zawarto kod źródłowy algorytmu typowania zasuw do zamknięcia z analizą przepływów (*Algorytm 3*).



Rys. 7.7 Schemat przepływu danych *Algorytmu 3* [opracowanie własne]

Ze względu na brak powyższych funkcjonalności w oprogramowaniu EPANET poprawność *Algorytmu 3* odpowiedzialnego za typowanie minimalnego zbioru zasuw do zamknięcia zweryfikowano na podstawie badań empirycznych. Badania składały się z analizy topologii sieci wodociągowej *ariel_phd.inp*, która została zamodelowana w Rozdziale 4.3 oraz przykładowego modelu hydraulicznego *Net3.inp*. Analiza wyników empirycznych potwierdziła poprawność działania zaproponowanego rozwiązania.

7.5 Algorytm klasyfikacji awarii

Celem procesu klasyfikacyjnego jest określenie wpływu poszczególnych awarii na całą infrastrukturę wodociągową oraz funkcjonowanie obszaru, na którym wystąpiły. Oznacza to, że proces klasyfikacji jest wysoce istotny w procesie szeregowania zadań, a jego poprawność uzależniona jest przede wszystkim od posiadanych informacji o SZZwW oraz typie awarii. Na podstawie przeglądu literatury oraz wiedzy eksperta odpowiedzialnego za nadzór ekip naprawczych określono trzy rodzaje awarii:

- (F1) – awaria niewymagająca przerwania działania procesu dystrybucji na analizowanym odcinku, ani wymiany przewodu;
- (F2) – awaria wymagająca przerwanie działania procesu dystrybucji wody na danym odcinku, ale niewymagająca wymiany uszkodzonego przewodu;
- (F3) – awaria wymagająca wymianę przewodu, w skutku którego konieczna jest przerwa w procesie dystrybucji wody w analizowanym odcinku.

Tab. 7.2 *Określenie przewidywanego czasu naprawy w zależności od typu awarii [10]*

Określenie czynności branych pod uwagę w procesie obliczania czasu T_{PN}				
Identyfikator [ID]	**Nazwa czynności**	**Rodzaj awarii**		
		F1	**F2**	**F3**
1	Otwarcie / Zamknięcie zasuwy	nie	tak	tak
2	Lokalizacja awarii	tak	tak	tak
3	Zbijanie nawierzchni asfaltowej	tak/nie	tak/nie	tak/nie
4	Ściąganie płyt chodnikowych	tak/nie	tak/nie	tak/nie
5	Wykop do rurociągu (mechaniczny)	tak	tak	tak
6	Odpompowanie wody z wykopu	tak/nie	tak/nie	tak/nie
7	Czyszczenie przewodu	nie	tak	tak
8	Montaż opaski naprawczej	tak	tak	nie
9	Montaż doszczelnienia	tak	tak	tak
10	Wycięcie odcinka przewodu	nie	nie	tak
11	Montaż nowego przewodu	nie	nie	tak
12	Otwarcie wody, odpowietrzenie, płukanie	nie	tak	tak
13	Zasypanie wykopu	tak	tak	tak
14	Czynności wykończeniowo-porządkowe	tak	tak	tak

Na podstawie informacji o rodzaju awarii możliwe będzie określenie przewidywanego czasu naprawy (T_{PN}), który stanowi główny wskaźnik determinujący jej klasę. W Tabeli 7.2 przedstawiono czynności brane pod uwagę w procesie obliczania prawdopodobnego czasu naprawy (T_{PN}). W sytuacji wystąpienia awarii typu (F1) pod uwagę nie bierze się czasu potrzebnego na otwarcie i zamknięcie zasuw, otwarcie wody (w tym odpowietrzenie i płukanie), czyszczenie przewodu, wycięcie przewodu czy montaż nowego przewodu. W przypadku awarii (F2) wycięcie i montaż nowego odcinka nie jest brany pod uwagę, a w przypadku awarii (F3) nie montuje się opaski naprawczej. Czynności takie jak zbijanie nawierzchni asfaltowej, ściąganie płyt chodnikowych uzależnione jest od miejsca wystąpienia awarii, natomiast odpompowanie wody

z wykopu uzależnione jest od czasu podjęcia rzeczywistych czynności naprawczych. Poszczególne czasy zostały zapisane w pliku konfiguracyjnym (*config.py*).

Celem algorytmu klasyfikacji awarii (dalej zwanego *Algorytmem 4*) jest przyporządkowanie zbioru zgłoszonych awarii do poszczególnych klas (C1, C2 lub C3). Klasyfikacja przebiega na podstawie podstawowych informacji o awarii takich jak: rodzaj awarii, miejsce wystąpienia, rodzaj nawierzchni oraz przewidywanego czasu naprawy (T_{PN}), procentowego udziału przewodu w transporcie wody (P_{DPW}), wskaźnika krytyczności przewodu (F_{KP}) oraz wskaźnika intensywności uszkodzeń (λ). W celu określenia klasy awarii opracowano prostą strukturę opisującą zbiór awarii, której przykład (dwie awarie) został przedstawiony poniżej:

```
{'failure_id': 'fail_1',
 'failure_type': 'F1',
 'pipe_id': 'P10',
 'pipe_diameter': None,
 'historical_data': None,
 'failure_start': '12:00:00',
 'type_of_substrate': None,
 'place_of_occurrence': None,
 'segment': None,
 'element_criticality': None,
 'damage_intensity_factor': None,
 'day_flow': None,
 'pipe_resilience_factor': None,
 'failure_time_repair': None,
 'failure_class': None,
 'prioritization_factor': False}
```

```
{'failure_id': 'fail_2',
 'failure_type': 'F3',
 'pipe_id': 'P23',
 'pipe_diameter': None,
 'historical_data': None,
 'failure_start': '12:00:00',
 'type_of_substrate': None,
 'place_of_occurrence': None,
 'segment': None,
 'element_criticality': None,
 'damage_intensity_factor': None,
 'day_flow': None,
 'pipe_resilience_factor': None,
 'failure_time_repair': None,
 'failure_class': None,
 'prioritization_factor': False}
```

Każda z awarii występujących w sieci wodociągowej musi zawierać identyfikator awarii (*failure_id*), rodzaj awarii (*failure_type*), identyfikator przewodu, który uległ awarii (*pipe_id*) oraz czas wystąpienia awarii (*failure_start*). Miejsce wystąpienia (*place_of_occurrence*) oraz rodzaj podłoża (*type_of_substrate*) są informacjami dodatkowymi, które użytkownik może uzupełnić. Jeśli użytkownik nie poda powyższych informacji, algorytm założy następujący scenariusz (najgorsza możliwość): miejsce prywatne oraz obecność płyt chodnikowych i asfaltu. Dane o średnicy przewodu (*pipe_diameter*) zostaną pobrane z modelu hydraulicznego, natomiast informacje o segmencie (*segment*) oraz klasyfikacji elementu sieci wodociągowej (*element_criticality*) zostaną uzupełnione automatycznie w oparciu o kolejno: *Algorytm 3* i metodę *find_segment* oraz *Algorytm 1* i metodę *network_resilience_algorithm*. Pozostałe informacje zostaną określone w oparciu o zrealizowany algorytm. Dane

o awariach pobierane są bezpośrednio z pliku o formacie .JSON, którego ścieżka dostępu znajduje się w pliku konfiguracyjnym. W *Załączniku G* niniejszej rozprawy umieszczono zawartość pliku z informacjami o analizowanych awariach w prezentowanym przykładzie. W celu określenia klas poszczególnych awarii konieczne jest utworzenie nowego obiektu przechowującego informacje o modelu hydraulicznym i zbiorze analizowanych awarii. Kolejnym krokiem jest wywołanie metody *update_data*, której zadaniem jest uzupełnienie podstawowych informacji o awariach. Przykład utworzenia obiektu i wywołanie metody aktualizującej dane przedstawiono poniżej:

```
# =========== Utworzenie obiektu testowego:
new_case = Algorithm_4()

# =========== Wywołanie metody aktualizującej dane:
new_case.update_data()
```

Wywołanie powyższej metody spowoduje aktualizację danych o awariach do następującej postaci (zmiany oznaczono kolorem czerwonym):

```
{'failure_id': 'fail_1',
 'failure_type': 'F1',
 'pipe_id': 'P10',
 'pipe_diameter': [24, 'D1'],
 'historical_data':
    {'number_of_failure': 10,
     'years': 4},
 'failure_start': '12:00:00',
 'type_of_substrate':
     ['asphalt', 'paving_slabs'],
 'place_of_occurrence': 'public',
 'segment':
     {'id': 'S1',
      'segment_links':
      ['P10', 'P2', 'P40', 'PUMP1',
       'PUMP_2', 'P1'],
      'segment_nodes':
      ['J6', 'J5', 'R1'],
      'near_node_list': ['J7','J1'],
      'segment_valves':
      ['V6', 'V7', 'V1']},
 'element_criticality': [73.51,
                         'CAT-3'],
 'damage_intensity_factor': None,
 'day_flow': None,
 'pipe_resilience_factor': None,
 'failure_time_repair': None,
 'failure_class': None,
 'prioritization_factor': False}
```

```
{'failure_id': 'fail_2',
 'failure_type': 'F3',
 'pipe_id': 'P23',
 'pipe_diameter': [12, 'D3'],
 'historical_data':
     {'number_of_failure': 4,
      'years': 10},
 'failure_start': '12:00:00',
 'type_of_substrate':
     ['paving_slabs'],
 'place_of_occurrence': 'public',
 'segment':
     {'id': 'S13',
      'segment_links':
      ['P23', 'P18', 'P24'],
      'segment_nodes': ['J15'],
      'near_node_list':
      ['J14', 'J9', 'J16'],
      'segment_valves':
      ['V19', 'V12', 'V21', 'V20']},
 'element_criticality': [99.14,
                         'CAT-1'],
 'damage_intensity_factor': None,
 'day_flow': None,
 'pipe_resilience_factor': None,
 'failure_time_repair': None,
 'failure_class': None,
 'prioritization_factor': False}
```

Po zaktualizowaniu informacji o awariach algorytm umożliwia wyliczenie wskaźników opisanych w Rozdziale 6.2 z wykorzystaniem poniższych metod:

```
# ========== Prawdopodobny czas naprawy:
new_case.failure_time_repair(failures)

# ========== Procentowy udział przewodu w transporcie wody:
new_case.day_flow(failures)

# ========== Wskaźnik krytyczności przewodu:
new_case.pipe_resilience_factor(failures)

# ========== Wskaźnik intensywności uszkodzeń:
new_case.damage_intensity_factor(failures)
```

Wywołanie powyższych metod spowoduje ponowne zaktualizowanie danych do postaci przedstawionej poniżej (zmiany oznaczono kolorem czerwonym):

```
{'failure_id': 'fail_1',
 'failure_type': 'F1',
 'pipe_id': 'P10',
 'pipe_diameter': [24, 'D1'],
 'historical_data':
    {'number_of_failure': 10,
     'years': 4},
 'failure_start': '12:00:00',
 'type_of_substrate':
     ['asphalt', 'paving_slabs'],
 'place_of_occurrence': 'public',
 'segment':
     {'id': 'S1',
      'segment_links':
      ['P10', 'P2', 'P40', 'PUMP1',
       'PUMP_2', 'P1'],
      'segment_nodes':
      ['J6', 'J5', 'R1'],
      'near_node_list': ['J7','J1'],
      'segment_valves':
      ['V6', 'V7', 'V1']},
 'element_criticality': [73.51,
                         'CAT-3'],
 'damage_intensity_factor': 0.0082,
 'day_flow': 18.9726,
 'pipe_resilience_factor': 1.331,
 'failure_time_repair':
         ['9 godz. 47 min', 587],
 'failure_class': None,
 'prioritization_factor': False}
```

```
{'failure_id': 'fail_2',
 'failure_type': 'F3',
 'pipe_id': 'P23',
 'pipe_diameter': [12, 'D3'],
 'historical_data':
     {'number_of_failure': 4,
      'years': 10},
 'failure_start': '12:00:00',
 'type_of_substrate':
     ['paving_slabs'],
 'place_of_occurrence': 'public',
 'segment':
     {'id': 'S13',
      'segment_links':
      ['P23', 'P18', 'P24'],
      'segment_nodes': ['J15'],
      'near_node_list':
      ['J14', 'J9', 'J16'],
      'segment_valves':
      ['V19', 'V12', 'V21', 'V20']},
 'element_criticality': [99.14,
                         'CAT-1'],
 'damage_intensity_factor': 0.0013,
 'day_flow': 0.9655,
 'pipe_resilience_factor': 0.324,
 'failure_time_repair':
         ['5 godz. 45 min', 345],
 'failure_class': None,
 'prioritization_factor': False}
```

Ostatnią metodą dostępną po zaktualizowaniu informacji o występujących awariach jest metoda *get_classify*, która na podstawie dostępnych danych dokonuje

klasyfikacji awarii na jedną z trzech dostępnych klas: (C_1), (C_2) lub (C_3). Sposób wnioskowania algorytmu przedstawiono poniżej:

- dodaj awarię do klasy (C1) w przypadku gdy dwa z poniższych warunków są prawdziwe:
  - uszkodzony element znajduje się w kategorii CAT-3 lub CAT-4;
  - dobowy przepływ w przewodzie stanowi co najmniej 15% sumarycznego przepływu we wszystkich przewodach;
  - przewidywany czas naprawy T_{PN} wynosi powyżej 15 godzin;
  - wskaźnik krytyczności przewodu jest powyżej 1.0;
- jeśli analizowana awaria nie spełniła co najmniej dwóch powyższych warunków to dodaj analizowaną awarię do klasy (C2) gdy dwa z poniższych warunków zostaną spełnione:
  - uszkodzony element znajduje się co najmniej w kategorii CAT-2;
  - dobowy przepływ w przewodzie stanowi co najmniej 10% sumarycznego przepływu we wszystkich przewodach;
  - przewidywany czas naprawy T_{PN} wynosi powyżej 8 godz. i 30 min;
  - wskaźnik krytyczności przewodu jest powyżej 0.5;
- jeśli analizowana awaria nie została przypisana do żadnej z powyższych klas, dodaj ją do klasy (C3).

Powyższe progi zostały opracowane na podstawie danych literaturowych i wiedzy eksperta dziedzinowego, natomiast można je dowolnie zmieniać wykorzystując plik konfiguracyjny. Wywołanie metody klasyfikującej zaprezentowano poniżej:

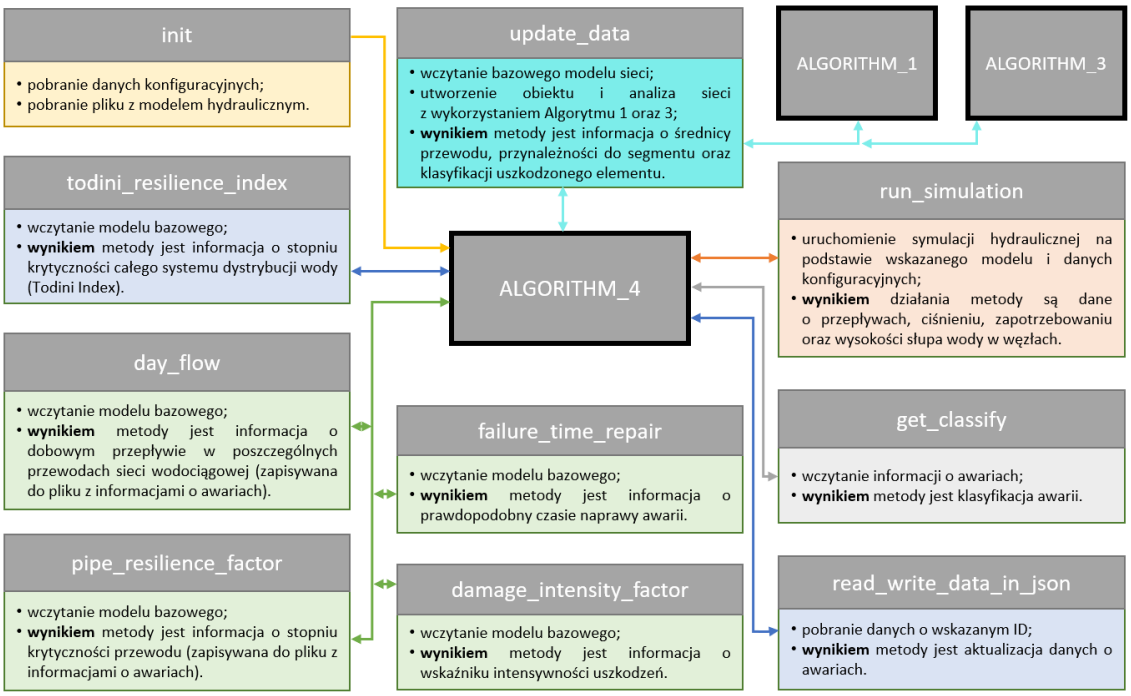
```
# =========== Klasyfikacja awarii:
new_case.get_classify(failures)
```

Wynikiem klasyfikacji przykładowego zbioru awarii jest kategoria (C1) dla awarii *fail_1*, oraz kategoria (C3) dla awarii o identyfikatorze *fail_2*.

Algorytm 4 umożliwia dodatkowo możliwość określenia zdolności systemu wodociągowego w oparciu o indeks Todiniego. W powyższym celu należy wywołać poniższą metodę [39, 40]:

```
# ========== Wywołanie metody Todini Index:
new_case.todini_resilience_index()
```

Indeks Todiniego dla analizowanej sieci wynosi 0.0405 co wskazuje na wysoką odporność na awarie. Na rys. 7.8 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku I* niniejszej rozprawy zawarto kod algorytmu typowania zasuw do zamknięcia z analizą przepływów (*Algorytm 4*).



Rys. 7.8 Schemat przepływu danych *Algorytmu 4* [opracowanie własne]

Poprawność *Algorytmu 4* odpowiedzialnego za klasyfikację awarii zweryfikowano na podstawie badań empirycznych. Badania składały się z analizy topologii sieci wodociągowej *ariel_phd.inp*, która została zamodelowana w Rozdziale 4.3 oraz przykładowego modelu hydraulicznego *Net3.inp*. Na podstawie 6 przypadków testowych

składających się z losowego zbioru awarii porównano wyniki klasyfikacji uzyskane przez *Algorytm 4* z klasyfikacją tego samego zbioru przez eksperta dziedzinowego. Na 150 awarii, 138 zostało zaklasyfikowane identycznie przez algorytm i eksperta, co stanowi 92% przypadków testowych. Analiza wyników empirycznych potwierdziła poprawność działania zaproponowanego rozwiązania.

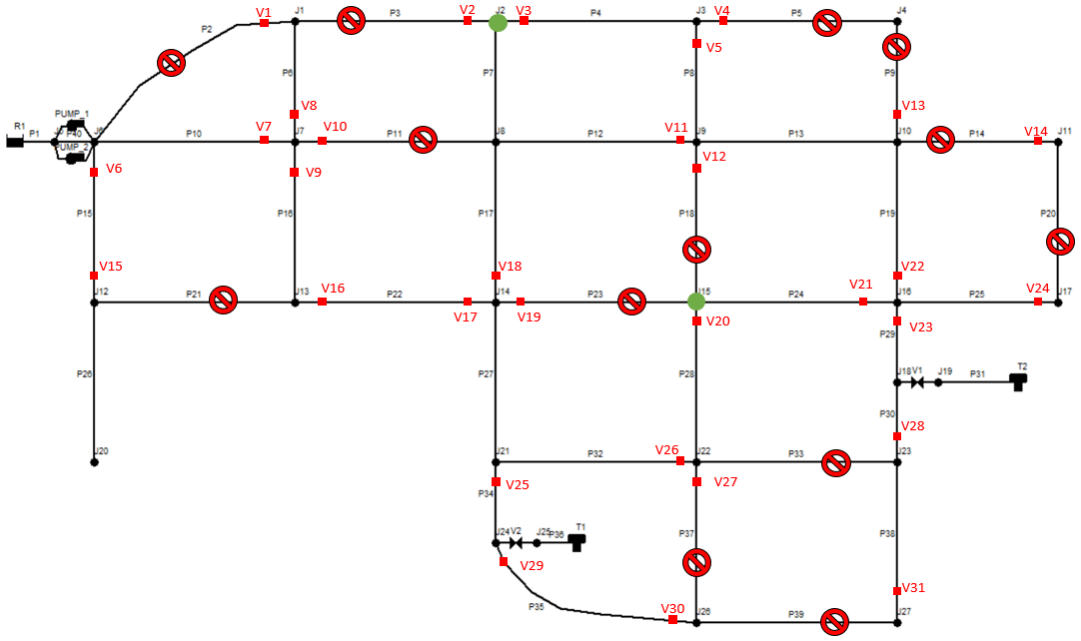
7.6 Algorytm priorytetyzacji

W przypadku sytuacji kryzysowej, prosta klasyfikacja awarii jest nie wystarczająca. Spowodowane jest to faktem, że określa ona jedynie wpływ awarii na proces dystrybucji wody i funkcjonowanie systemu zbiorowego zaopatrzenia w wodę. Proces klasyfikacji nie uwzględnia natomiast zachowania infrastruktury oraz obszaru, w którym sieć funkcjonuje. Niezmiernie ważne jest, aby w takich sytuacjach uwzględniać takie elementy jak infrastruktura krytyczna[7] (np. szpitale), która powinna posiadać stały dostęp do zasobów wodnych czy energetycznych. Powyższy fakt implikuje konieczność wprowadzenia mechanizmu priorytetyzacji. Oznacza to, że podczas procesu decyzyjnego ustalając kolejność naprawy awarii należy wziąć pod uwagę ich ważność. Na podstawie informacji uzyskanych przez eksperta dziedzinowego zdecydowano się na wprowadzenie binarnej priorytetyzacji awarii (True / False). Celem algorytmu priorytetyzacji awarii (dalej zwanego *Algorytmem 5*) jest określenie na podstawie symulacji hydraulicznych, trasy przepływu wody (*Algorytm 2*) oraz klasyfikacji awarii (*Algorytm 4*), czy wśród zgłoszonych awarii istnieją takie, które stanowią główny szlak dystrybucji wody do infrastruktury krytycznej.

W celu omówienia działania *Algorytmu 5* założono wystąpienie sytuacji kryzysowej w mieście reprezentowanym przez model hydrauliczny *ariel_phd.inp*. Przypadek testowy zawiera losowo wybrane miejsca i typ uszkodzeń. Miasto posiada dwa budynki infrastruktury krytycznej, które czerpią wodę kolejno z węzła J2 oraz J15.

[7] Zgodnie z definicją przedstawioną w ustawie o zarządzaniu kryzysowym z dnia 26 kwietnia 2007 roku (DzU z 2007r. nr 89 poz. 590, ze zm.) poprzez infrastrukturę krytyczną rozumie się *systemy oraz wchodzące w ich skład powiązane ze sobą funkcjonalne obiekty, w tym obiekty budowlane, urządzenia, instalacje, usługi kluczowe dla bezpieczeństwa państwa i jego obywateli oraz służące zapewnieniu sprawnego funkcjonowania organów administracji publicznej, a także instytucji i przedsiębiorców.*

Informacje o typie i lokalizacji awarii w analizowanym przykładzie umieszczono w *Załączniku G* niniejszej rozprawy, natomiast informację o identyfikatorach węzłów, które dostarczają wodę do budynków infrastruktury krytycznej w *Załączniku H*. Na podstawie tych danych na rys. 7.9 przedstawiono rozmieszczenie awarii i węzłów infrastruktury krytycznej.



Rys. 7.9 Przykładowy przypadek testowy [opracowanie własne]

Pierwszym etapem koniecznym do uruchomienia algorytmu jest utworzenie obiektu reprezentującego scenariusz testowy. Analogicznie jak w przypadku algorytmów opisanych we wcześniejszych rozdziałach, jego obsługa polega na wywołaniu odpowiednich metod zaimplementowanych w języku Python. Ścieżki dostępu do modelu hydraulicznego, informacji o infrastrukturze krytycznej, podstawowych danych o awariach oraz czasu trwania symulacji pobierane są z pliku konfiguracyjnego (*config.py*). Wywołanie metody utworzenia obiektu oraz metody pobierającej dane o awariach zostało zaprezentowane poniżej:

```
# =========== Utworzenie obiektu:
new_case = Algorithm_5()

# =========== Aktualizacja danych o awariach:
new_case.get_initial_data()
```

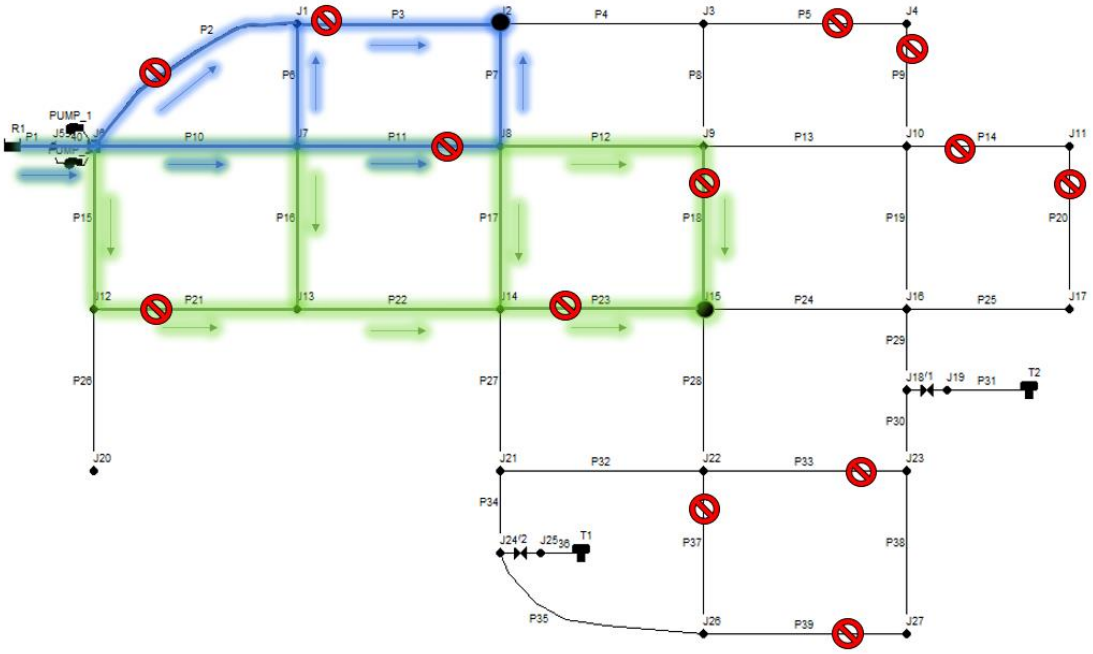
Na podstawie informacji o awariach algorytm odrzuca wszystkie te, które określono typem (F1), ponieważ ich naprawa nie skutkuje odcięciem segmentu, a co za tym idzie nie może spowodować sytuacji blokującej szlak wody od źródła do obiektu infrastruktury krytycznej. Następnie po uproszczeniu listy awarii, algorytm dokonuje ich klasyfikacji za pośrednictwem *Algorytmu 4*. Informację o awariach, przynależności do segmentu oraz wynik ich klasyfikacji umieszczono w Tabeli 7.3.

Tab. 7.3 *Przykład awarii oraz ich klasyfikacja w przykładowym modelu testowym [opracowanie własne]*

Informacje o awariach w modelu *ariel_phd.inp*							
ID	**TYP**	**ID SEGMENTU**	**KATEGORIA**	**WSKAŹNIK KRYTYCZNOŚCI PRZEWODU**	**BILANS DOBOWEGO PRZEPŁYWU**	**T_{PN}**	**KLASA AWARII**
fail_P2	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P3	F2	S5	CAT-1	0,104	0,02	08:24:00	C3
fail_P5	F2	S9	CAT-1	0,273	0,51	08:16:00	C3
fail_P9	F3	S9	CAT-1	0,222	0,01	09:33:00	C3
fail_P11	F2	S6	CAT-3	1,593	36,19	17:52:00	C1
fail_P14	F2	S10	CAT-1	0,362	2,19	09:45:00	C3
fail_P18	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P20	F3	S11	CAT-1	0,304	0,03	08:42:00	C3
fail_P21	F2	S3	CAT-1	0,253	0,04	08:21:00	C3
fail_P23	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P33	F2	S15	CAT-1	0,510	0,79	13:33:00	C2
fail_P37	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P39	F2	S19	CAT-1	0,241	0,31	08:11:00	C3

Kolejnym etapem jest określenie śladu wody od źródła do węzła, z którego czerpie wodę obiekt infrastruktury krytycznej. Na podstawie informacji o tzw. węzłach krytycznych (J2 i J15) algorytm priorytetyzacji wykorzystuje metodę *path_creator* algorytmu odpowiedzialnego za wyznaczanie tras przepływu wody (*Algorytm 2*). Na podstawie wyników *Algorytm 5* dokonuje weryfikacji, czy uszkodzone przewody znajdują się na trasie między źródłami infrastruktury krytycznej.

Rezultatem działania algorytmu jest lista awarii, które znajdują się na głównym szlaku dystrybucji wody do punktu krytycznego. W analizowanym przykładzie są to awarie o identyfikatorach: *fail_P3*, *fail_P11* oraz *fail_P21*. Na rys. 7.10 przedstawiono trasy przepływów wody (kolorem niebieskim od źródła R1 do węzła J2, natomiast kolorem zielonym od źródła R1 do węzła J15) oraz lokalizacje awarii.



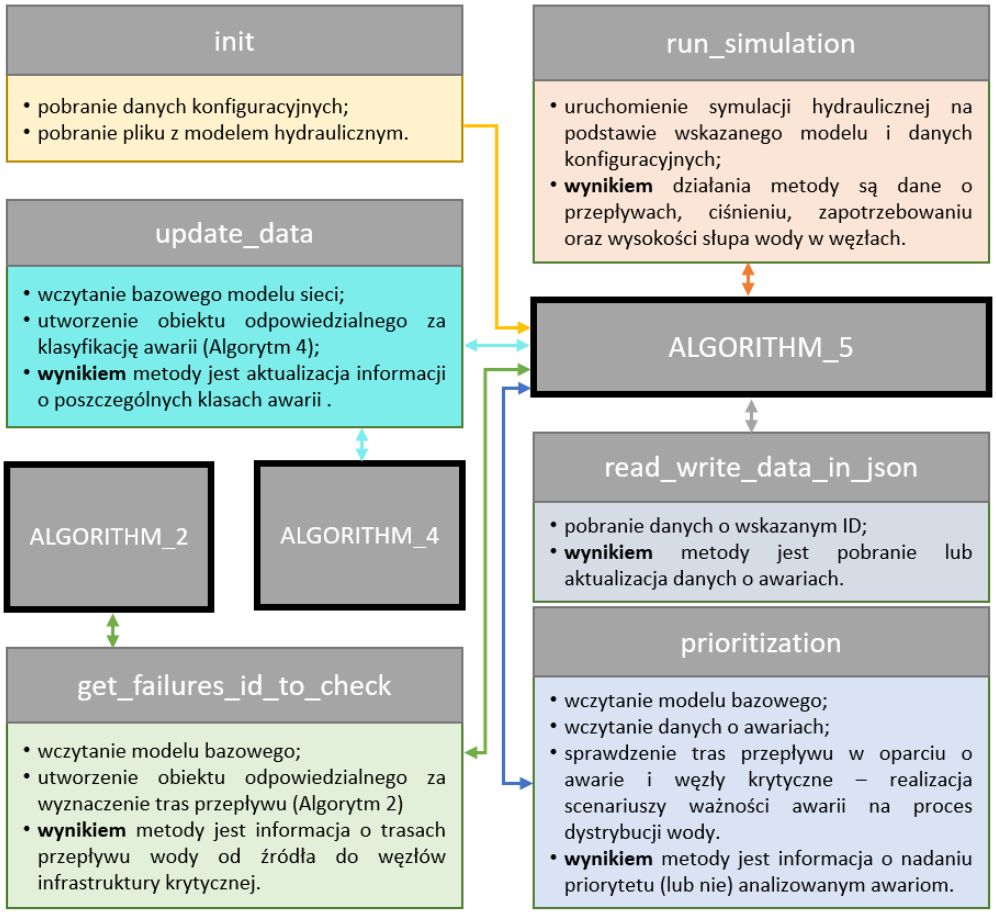
Rys. 7.10 Trasy przepływów wody do węzłów krytycznych wraz z lokalizacją awarii [opracowanie własne]

Ostatnim krokiem *Algorytmu 5* jest utworzenie modeli hydraulicznych i sprawdzenie za pośrednictwem metody *prioritization*, które awarie powodują brak dostępu wody w węzłach krytycznych. Algorytm w sposób równoległy sprawdza wszystkie możliwe kombinacje. Przykład wywołania powyższej metody przedstawiono poniżej:

```
# ========== Wywołanie metody odpowiedzialnej za
# ========== priorytetyzację:
new_case.prioritization()
```

Okazuje się, że w przypadku węzła J2 priorytet powinna otrzymać awaria *fail_P3* lub *fail_P11*, natomiast w przypadku węzła J15, awaria *fail_P11*. Analizując oba przypadki jednocześnie, priorytet należy przyznać awarii *fail_P11*, ze względu na występowanie w obu szlakach transportujących wodę do węzłów krytycznych (awaria została oznaczona w Tabeli 7.3 kolorem czerwony). W sytuacji gdy wiele awarii posiada priorytet, kolejność ich określana jest na podstawie przynależności do klas (C1, C2, C3) oraz procentowym udziale przewodu w dobowym przepływie całej sieci.

Na rys. 7.11 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku J* niniejszej rozprawy zawarto kod algorytmu priorytetyzacji (*Algorytm 5*).



Rys. 7.11 Schemat przepływu danych *Algorytmu 5* [opracowanie własne]

Poprawność *Algorytmu 5* odpowiedzialnego za priorytetyzację awarii zweryfikowano na podstawie badań empirycznych. Badanie składało się z analizy topologii sieci wodociągowej *ariel_phd.inp*, która została zamodelowana w Rozdziale 4.3 oraz przykładowego modelu hydraulicznego *Net3.inp*. Na podstawie 10 przypadków testowych składających się z losowego zbioru awarii porównano wyniki priorytetyzacji uzyskane przez *Algorytm 5* z priorytetyzacja tego samego zbioru przez eksperta dziedzinowego. Na 164 awarie, ekspert nadał priorytet 15 awariom, natomiast algorytm 16. Zbiór awarii z priorytetem nadanym przez eksperta pokrywał się w 93.3% ze zbiorem zaproponowanym przez algorytm. Po przeanalizowaniu różnicy ekspert zgodził się

z rozwiązaniem algorytmu. Niebywałą zaletą prezentowanego podejścia jest również czas, ponieważ ekspert wykonał pracę priorytetyzacji w około 6 godzin, gdy algorytm zaproponował swoje rozwiązania łącznie po około 24 minutach. Analiza wyników empirycznych potwierdziła poprawność działania zaproponowanego rozwiązania.

7.7 Algorytm agregacji awarii

Głównym celem wprowadzenia możliwości agregacji awarii jest minimalizacja sumarycznej liczby zasuw niezbędnych do zamknięcia, w celu przywrócenia ciągłości procesu dystrybucji wody. W sytuacjach kryzysowych, gdzie liczba awarii jest szczególnie wysoka istnieje wysokie prawdopodobieństwo wystąpienia w jednym segmencie wielu awarii. W takich sytuacjach należałoby zastanowić się, czy istnieje możliwość naprawy tych awarii odcinając segment tylko jeden raz. Z drugiej strony warto zwrócić uwagę na sytuację wystąpienia awarii w segmentach sąsiadujących ze sobą (posiadających wspólną zasuwę). W takiej sytuacji można potraktować oba segmenty jako jeden wspólny, czego efektem jest również zmniejszona sumaryczna liczba zasuw do zamknięcia. Celem algorytmu agregacji awarii (dalej zwanego *Algorytmem 6*) jest określenie na podstawie symulacji hydraulicznych, czy wśród zgłoszonych awarii istnieją takie, które posiadają wspólne zasuwy do zamknięcia (algorytm nie analizuje awarii o wysokim priorytecie).

Analogicznie do sposobu obsługi algorytmów opisanych powyżej, prezentowany algorytm również wykorzystuje metody zaimplementowane w języku Python. W celu określenia możliwych agregacji należy w pliku bazowym algorytmu utworzyć nowy obiekt. Ścieżka dostępu do modelu hydraulicznego i czas trwania symulacji pobierana jest z pliku konfiguracyjnego (*config.py*). Po utworzeniu obiektu należy wywołać metodę aktualizującą dane o awariach (algorytm klasyfikacyjny – *Algorytm 4*) oraz uruchomić metodę priorytetyzacji (*Algorytm 5*). Po uzupełnieniu niezbędnych informacji dotyczących klasyfikacji i priorytetyzacji algorytm umożliwia wywołanie dwóch metod:

- *agregation_first_deegre*, której zadaniem jest wyszukiwanie awarii typu F2 i F3 w obrębie jednego segmentu;
- *agregation_secondt_deegre*, której zadaniem jest wyszukiwanie awarii typu F2 i F3 w obrębie segmentów sąsiadujących.

Informacja o maksymalnym stopniu agregacji tj. maksymalnej liczbie awarii, które można złączyć w jedną, umieszczono w pliku konfiguracyjnym (*config.py*). Wywołanie powyższych metod przedstawia się następująco:

```
# ========== Utworzenie nowego obiektu:
new_case = Algorithm_6()

# ========== Aktualizacja danych:
new_case.update_data()

# ========== Agregacja pierwszego stopnia:
new_case.agregation_first_deegre()

# ========== Agregacja drugiego stopnia:
new_case.agregation_second_deegre()
```

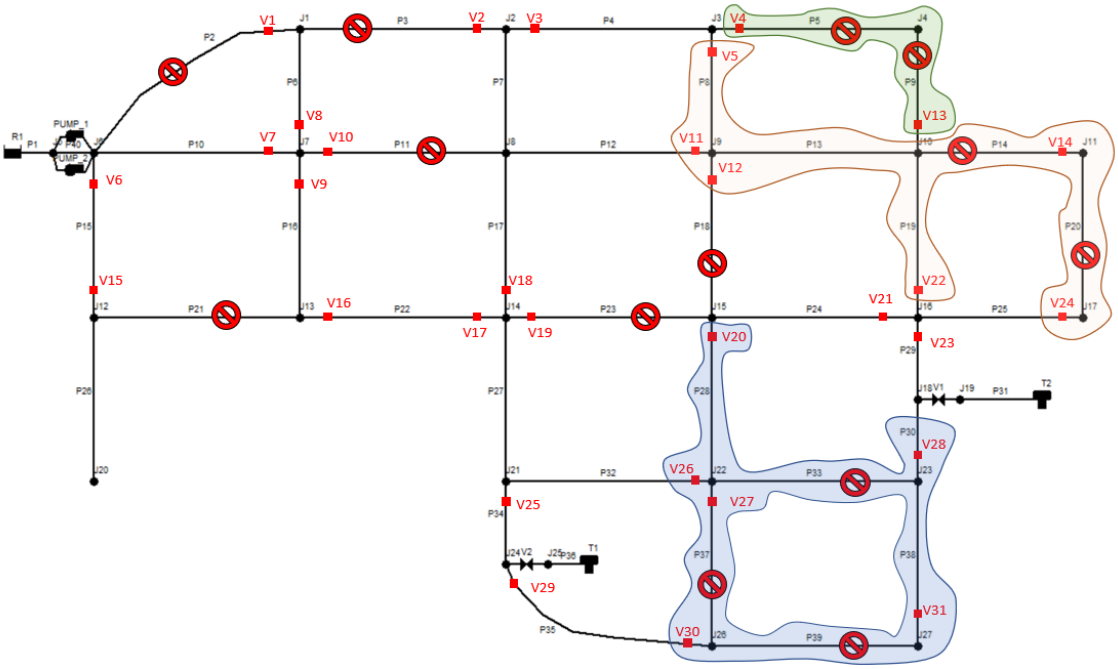
W celu omówienia działania *Algorytmu 6* przeanalizowano przypadek wystąpienia sytuacji kryzysowej opisanej w Rozdziale 7.6. W Tabeli 7.4 umieszczono informację o awariach, przynależności do segmentu, wyniku klasyfikacji i priorytetyzacji. Awarię *fail_P11*, która jest awarią priorytetową nie wzięto pod uwagę w procesie agregacji.

Tab. 7.4 *Przykład awarii oraz ich klasyfikacja w przykładowym modelu testowym [opracowanie własne]*

Informacje o awariach w modelu ariel_phd.inp							
ID	**TYP**	**ID SEGMENTU**	**KATEGORIA**	**WSKAŹNIK KRYTYCZNOŚCI PRZEWODU**	**BILANS DOBOWEGO PRZEPŁYWU**	**T_{PN}**	**KLASA AWARII**
fail_P2	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P3	F2	S5	CAT-1	0,104	0,02	08:24:00	C3
fail_P5	F2	S9	CAT-1	0,273	0,51	08:16:00	C3
fail_P9	F3	S9	CAT-1	0,222	0,01	09:33:00	C3
fail_P11	F2	awaria odrzucona przez algorytm ze względu na jej priorytet					
fail_P14	F2	S10	CAT-1	0,362	2,19	09:45:00	C3
fail_P18	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P20	F3	S11	CAT-1	0,304	0,03	08:42:00	C3
fail_P21	F2	S3	CAT-1	0,253	0,04	08:21:00	C3
fail_P23	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P33	F2	S15	CAT-1	0,510	0,79	13:33:00	C2
fail_P37	F1	awaria odrzucona przez algorytm ze względu na typ awarii (F1)					
fail_P39	F2	S19	CAT-1	0,241	0,31	08:11:00	C3

Wynik działania obu metod zaznaczono w Tabeli 7.4. Kolorem zielonym oznaczono awarie (*fail_P5*, *fail_P9*), które za pośrednictwem pierwszej metody

(*agregation_first*_degree) zostały wytypowane do naprawy w ramach jednego zlecenia. Sytuacja tego typu spowoduje brak konieczności powtórnego zamykania i otwierania tych samych zasuw, co w sytuacji krytycznej ma duże znaczenie. Kolorem czerwonym oraz niebieskim oznaczono natomiast awarie (*fail_P14*, *fail_P20* oraz *fail_P33*, *fail_P39*), które są awariami znajdującymi się w sąsiadujących segmentach. Algorytm zaleca naprawę wskazanych awarii w ramach jednego zlecenia, ponieważ zminimalizuje to liczbę zasuw do zamknięcia, co skróci sumaryczny czas naprawy obu awarii. Rys. 7.12 przedstawia rezultat działania algorytmu proponującego agregację awarii w postaci graficznej.

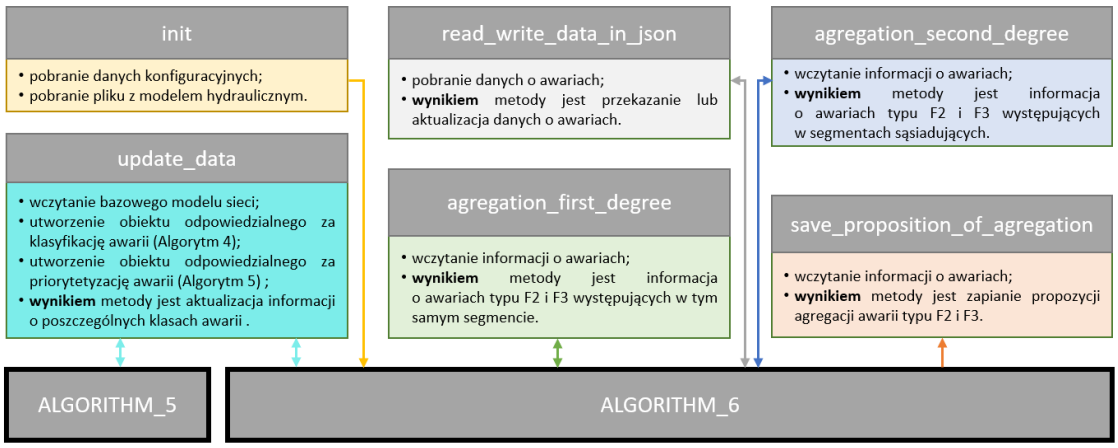


Rys. 7.12 Wynik działania Algorytmu 6 – agregacja awarii [opracowanie własne]

Ostatnią dostępną metodą w *Algorytmie* 6 jest możliwość zapisania propozycji agregacji do pliku o rozszerzeniu .JSON. Wywołanie metody *save_proposition _of_aggregation* przedstawiono poniżej:

```
# ========== Zapisanie wyników działania algorytmu:
new_case.save_proposition_of_aggregation()
```

Na rys. 7.13 przedstawiono schemat przepływu danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku K* niniejszej rozprawy zawarto kod źródłowy algorytmu agregacji awarii (*Algorytm 6*).



Rys. 7.13 Schemat przepływu danych *Algorytmu 6* [opracowanie własne]

Poprawność implementacji *Algorytmu 6* odpowiedzialnego za agregację awarii zweryfikowano na podstawie badań empirycznych. Badania składały się z analizy topologii sieci wodociągowej *ariel_phd.inp*, która została zamodelowana w Rozdziale 4.3 oraz przykładowego modelu hydraulicznego *Net3.inp*. Analiza wyników empirycznych potwierdziła poprawność działania zaproponowanego rozwiązania.

7.8 Algorytm szeregowania zadań dostępnych ekip remontowych

W sytuacjach katastroficznych, w których narażone są systemy zbiorowego zaopatrzenia w wodę, algorytmy zaproponowane w Rozdziałach 7.2 – 7.7 stanowią narzędzie umożliwiające klasyfikację elementów wchodzących w skład sieci oraz awarii, które w niej wystąpiły. Dodatkowo na podstawie wiedzy o lokalizacji zasuw oraz budynkach infrastruktury krytycznej algorytmy potrafią zaproponować agregację awarii oraz wskazać te awarie, które powinny zostać naprawione w pierwszej kolejności. Ostatnim elementem niezbędnym do osiągnięcia celu, jakim jest poprawa efektywności procesu przywracania ciągłości dystrybucji wody, są informację o liczbie i lokalizacji dostępnych ekip remontowych, materiale i narzędziach dostępnych w obrębie każdej z ekip oraz lokalizacji i stanie magazynów. Na podstawie powyższych danych określić

będzie można kolejność realizacji zleceń naprawczych wraz z przypisaniem identyfikatora ekipy odpowiedzialnej za powierzone zadanie. Celem algorytmu szeregowania zadań dostępnych ekip naprawczych (dalej zwanego *Algorytmem 7*) jest określenie kolejności napraw awarii na podstawie priorytetyzacji, klasyfikacji, stanu magazynowego, dostępności i czasu dojazdu ekip naprawczych. Ze względu na trudny dostęp do modeli hydraulicznych reprezentujących rzeczywiste obiekty algorytm został zaimplementowany w dwóch wersjach. Pierwsza wersja przeznaczona jest dla modeli nierzeczywistych, w których topologia sieci będzie określać również topologię ulic oraz wersja druga (dla modeli rzeczywistych) bazująca na dostępnych mapach drogowych on-line. Informacje o rodzaju analizowanego modelu (*real* lub *unreal*) należy umieścić w pliku konfiguracyjnym symulatora.

Kategoryzacja oraz priorytetyzacja awarii umożliwia wprowadzenie hierarchii kolejności typowania zleceń naprawczych. Na podstawie dostępnych kategorii zdecydowano się na 4 poziomy.

1. Poziom (L1) – krytyczny (natychmiastowy), w którym znajdują się wyłącznie awarie priorytetowe. Kolejność awarii na tym poziomie określa dodatkowo klasyfikator awarii (C1, C2, C3). W przypadku awarii o tej samej kategorii kolejność naprawy nie ma znaczenia.
2. Poziom (L2) – wysoki, w którym znajdują się kategorie klasy C1. Kolejność napraw awarii na tym poziomie nie ma znaczenia.
3. Poziom (L3) – średni zawierający zbiór kategorii C2. Kolejność napraw awarii na tym poziomie nie ma znaczenia.
4. Poziom (L4) – niski, w którym znajdują się kategorie klasy C3.

W przypadku agregacji awarii przynależących do różnych poziomów, awaria zagregowana przyjmuje poziom awarii wyższej klasy.

W celu określenia kolejności wykonywania napraw przez poszczególne ekipy remontowe konieczne jest utworzenie nowego obiektu przechowującego informacje o modelu hydraulicznym, zbiorze analizowanych awarii, dostępnych ekipach i ich lokalizacji oraz stanie i rozmieszczeniu magazynów. Kolejnym krokiem jest aktualizacja

danych o awariach wykorzystując metodę *update_data*. Przykład utworzenia obiektu i wywołanie metody aktualizującej dane przedstawiono poniżej:

```
# ========== Utworzenie obiektu testowego:
new_case = Algorithm_7()

# ========== Wywołanie metody aktualizującej dane:
new_case.update_data()
```

Po zaktualizowaniu wszystkich informacji należy utworzyć cztery poziomy zleceń naprawczych (L1) – (L4), do których algorytm przypisze określone identyfikatory awarii. Metodą odpowiedzialną za powyższe zadanie jest metoda *create_leves*, której sposób wywołania przedstawiono poniżej:

```
# ========== Wywołanie metody tworzącej poziomy L1 – L4 dla
# ========== awarii z uwzględnieniem agregacji:
new_case.create_levels(aggregation=True)

# ========== Wywołanie metody tworzącej poziomy L1 – L4 dla
# ========== awarii bez uwzględnieniem agregacji:
new_case.create_levels(aggregation=False)
```

Metoda *create_levels* umożliwia wykonanie procesu selekcji wykorzystując propozycje agregacji z *Algorytmu 6* lub traktując wszystkie awarie jako odrębne byty. W Tabeli 7.5 przedstawiono wyniki obu podejść (zmiany oznaczono kolorem zielonym oraz czerwonym).

Tab. 7.5 *Wyniki utworzenia poziomów awarii dla opisywanego przykładu [opracowanie własne]*

Klasyfikacja awarii do poszczególnych poziomów zleceń		
POZIOM ZLECEŃ	**TRYB**	**IDENTYFIKATORY AWARII**
L1	bez agregacji	*fail_P11*
	z agregacją	*fail_P11*
L2	bez agregacji	*fail_P18*
	z agregacją	*fail_P18*
L3	bez agregacji	*fail_P3, fail_P5, fail_P9, fail_P14, fail_P20, fail_P21, fail_P33, fail_P37, fail_P39*
	z agregacją	*fail_P3, fail_P21, fail_P37, AG_fail_P5_P9, AG_fail_P14_P20, AG_fail_P33_P39*
L4	bez agregacji	*fail_P2, fail_P23*
	z agregacją	*fail_P2, fail_P23*

Warto nadmienić, że w przypadku agregacji awarii algorytm wylicza nowe prawdopodobne czasy naprawy (T_{PN}). Spowodowane jest to brakiem konieczności wliczania ponownego czasu otwierania/zamykania zasuw. Powyższa metoda odpowiada dodatkowo za ustrukturyzowanie danych w taki sposób aby kolejne metody *Algorytmu 7* mogły analizować dane przechowywane w jednym miejscu. W trakcie tworzenia poziomów zleceń i przypisywania do nich awarii, każdej awarii przypisano zestaw następujących informacji: identyfikator awarii (*failure_id*), kategorię awarii (*failure_category*), identyfikator uszkodzonego przewodu (*pipe_id*), identyfikator węzła początkowego uszkodzonego przewodu (*start_node*_id) wraz ze współrzędnymi (*coordinates*), typ awarii (*failure_type*) oraz zbiór zasuw niezbędnych do odizolowania odcinka wraz z przypisanym identyfikatorem węzła i współrzędnymi. Przykład opisu awarii przedstawiono poniżej:

```
# ========== Przykład opisu awarii:
{  'failure_id': 'fail_P21',
   'failure_category': 'C2',
   'pipe_id': 'P21',
   'start_node': {'id': 'J12', 'coordinates': [10.0, 60.0]},
   'failure_type': 'F2',
   'valves_id': {'segment_id':
                {'segment_id': 'S12',
                'valves': [
                   {'valve_id': 'V15', 'node_id': 'J12',
                              'coordinates': [10.0, 62.0]},
                   {'valve_id': 'V16', 'node_id': 'J13',
                              'coordinates': [37.0, 60.0]},
                   {'valve_id': 'V9', 'node_id': 'J7',
                          'coordinates': [35.0, 78.0]}]}}}
```

Kolejnym elementem branym pod uwagę w procesie szeregowania zadań jest lokalizacja magazynów przechowujących niezbędny sprzęt i materiały. W trakcie procesu realizacji i implementacji algorytmu zdecydowano się na trzy elementy naprawcze:

- zaciski (ang. *clamps*) – służące do naprawy awarii typu F1 i F2, występujące w trzech różnych rozmiarach (D1, D2, D3);
- rury (ang. *pipes*) – wymagane w przypadku wymiany odcinka uszkodzonego rurociągu (awaria typu: F3), występujące w trzech różnych rozmiarach nawiązujących do średnicy przewodów (D1, D2, D3);

- inne narzędzia (ang. *other stuff*) – wymagane w przypadku naprawy wszystkich typów awarii (F1, F2, F3).

Informacje o ścieżce dostępu do pliku w formacie .JSON z danymi dotyczącymi lokalizacji magazynów oraz ich stanu pobierane są z pliku konfiguracyjnego *config.py*. W *Załączniku H* niniejszej rozprawy umieszczono zawartość pliku z powyższymi informacjami.

W celu zamodelowania sytuacji przedstawiających wykorzystanie magazynów i zużycie sprzętu zdecydowano, że do naprawy awarii:

- typu F1 należy zużyć: 2 zaciski (rozmiar uzależniony od średnicy przewodu) oraz 10 innych elementów (*other_stuff*);
- typu F2 należy zużyć: 4 zaciski (rozmiar uzależniony od średnicy przewodu) oraz 15 innych elementów (*other_stuff*);
- typu F3 należy zużyć: 2 przewody o odpowiedniej średnicy oraz 20 innych elementów (*other_stuff*).

Powyższe zależności można zmodyfikować w pliku konfiguracyjnym (*config.py*). W celu wprowadzenia większego zróżnicowania dodano zmienną określającą prawdopodobieństwo zmiany niezbędnych elementów naprawczych, która domyślnie ustawiona jest na 15 procent. Oznacza to, że w sytuacji gdy ekipa naprawcza dotrze do miejsca naprawy awarii istnieje prawdopodobieństwo zmiany liczby wymaganego sprzętu do naprawy. Jeżeli ekipa nie będzie posiadać odpowiedniej liczby materiałów – konieczne będzie uzupełnienie materiałów w najbliższym magazynie (lub zmiana dyspozycji na inną awarię). Przykład wywołania metody zliczającej prawdopodobne całkowite zużycie materiałów i sprzętu przedstawiono poniżej:

```
# ========== Wywołanie metody obliczającej koszt materiałów:
new_case.failure_reapir_cost()
```

Powyższa metoda nie tylko obliczy prawdopodobne całkowite zużycie materiałów, ale uzupełni również opis awarii o dane dotyczące wymagań materiałowych i sprzętowych. Wiedza ta pozwoli sprawdzić czy stan magazynowy lub ekipa otrzymująca zlecenie

naprawy posiada wystarczającą liczbę materiałów. Należy dodać, że magazyny zostały zaprojektowane w taki sposób, aby przechowywały losową liczbę materiałów na starcie symulacji, wskazując ich maksymalną pojemność i minimalny stopień wypełnienia podawany w procentach. W Tabeli 7.6 zestawiono informacje dotyczące występujących awarii oraz ich wymagań materiałowych.

Tab. 7.6 *Wymagania materiałowo-sprzętowe w celu naprawy awarii [opracowanie własne]*

Wymagania materiałowo-sprzętowe niezbędne do naprawy awarii			
IDENTYFIKATOR AWARII	**ZACISKI**	**PRZEWODY**	**INNE NARZĘDZIA**
fail_P11	*4*	*0*	*15*
fail_P18	*2*	*0*	*10*
fail_P33	*4*	*0*	*15*
fail_P3	*4*	*0*	*15*
fail_P21	*4*	*0*	*15*
fail_P37	*2*	*0*	*10*
fail_P39	*4*	*0*	*15*
AG_fail_P5_P9	*4*	*2*	*35*
AG_fail_P14_P20	*4*	*2*	*35*
fail_P2	*2*	*0*	*10*
fail_P23	*2*	*0*	*10*
RAZEM	**36**	**4**	**185**

W sytuacji, gdy w magazynach będzie brakować materiałów algorytm przeanalizuje, które awarie można naprawić w czasie oczekiwania na dostawę sprzętu do magazynu. Awarie, które z powyższego powodu nie mogą zostać naprawione zostają zawieszone na określony w pliku konfiguracyjnym czas (*suspended_time*). Aby uniknąć sytuacji, w której materiały zamawiane są dopiero w momencie ich braku algorytm co zadeklarowany czas (*delivery_time*) będzie w sposób losowy powiększać liczebność materiałów w magazynie.

Ostatnim elementem niezbędnym w procesie szeregowania zadań ekip naprawczych jest informacja o liczbie i lokalizacji ekip. Na potrzeby przeprowadzania różnych scenariuszy testowych zdecydowano się na utworzenie osobnego pliku o formacie .JSON zawierającego tego typu dane (dane o ścieżce dostępu zawarte są w pliku konfiguracyjnym). W przypadku pierwszej wersji algorytmu poprzez lokalizację ekip rozumie się identyfikator węzła, natomiast w wersji drugiej rzeczywiste współrzędne

geograficzne. Poza lokalizacją początkową ekip naprawczych w pliku przechowywane są również informacje o ładowności pojazdów oraz liczbie poszczególnych materiałów naprawczych. Fragment pliku przedstawiono poniżej:

```
# =========== Fragment pliku przechowujący informację
# =========== o ekipach naprawczych:
{
    "network_type": "unreal",
    "teams": [{
        "id": "RT1",
        "localisation": "J2",
        "max_capacity": {
            "clamps": 13,"pipes": 2,"other_stuff": 40
        },
        "stock_levels": {
            "clamps": {
                "D1": 5, "D2": 4, "D3": 4
            },
            "pipes": {
                "D1": 1,"D2": 0, "D3": 1
            },
            "other_stuff": 30
        }},
…
```

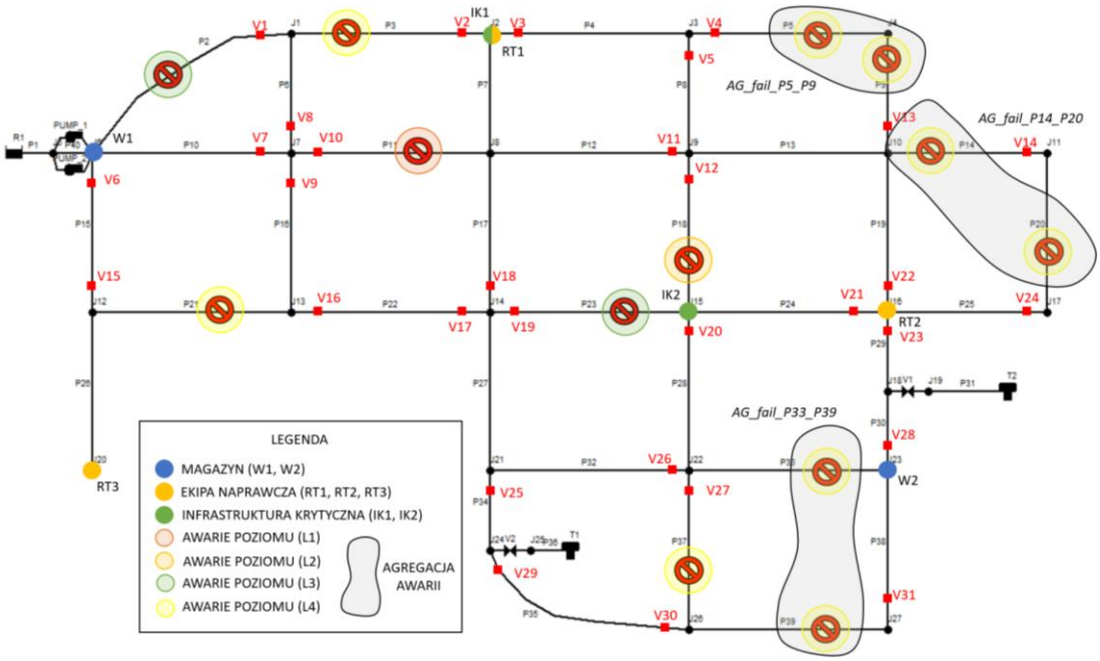
Na podstawie powyższych danych można przygotować scenariusz, który zostanie poddany analizie przez algorytm. W celu przygotowania scenariusza należy wywołać metodę *create_scenario*, której przykład wywołania przedstawiono poniżej:

```
# =========== Wywołanie metody tworzącej scenariusz testowy:
new_case.create_scenario()
```

Przykład wywołania metody tworzącej scenariusz spowoduje wywołanie wcześniej opisanych algorytmów (*Algorytm 1 – 6*), których wyniki są niezbędne do przeprowadzenia procesu szeregowania zadań. Wynikiem działania metody jest kompletny zestaw danych, na podstawie których *Algorytm 7* będzie w stanie określić kolejność wykonywanych napraw i ekip, które będą odpowiadać za naprawę awarii.

W celu omówienia działania *Algorytmu 7* założono wystąpienie sytuacji kryzysowej w mieście reprezentowanym przez model hydrauliczny *ariel_phd.inp* zaprezentowany w poprzednich rozdziałach. Dodatkowo założono obecność trzech ekip remontowych RT1, RT2 oraz RT3 oraz dwóch magazynów W1 oraz W2. Zawartość pliku

z analizowanym scenariuszem umieszczono w *Załączniku L* niniejszej pracy. Rys. 7.14 przedstawia w formie graficznej rezultat wywołania metody *create_scenario*.

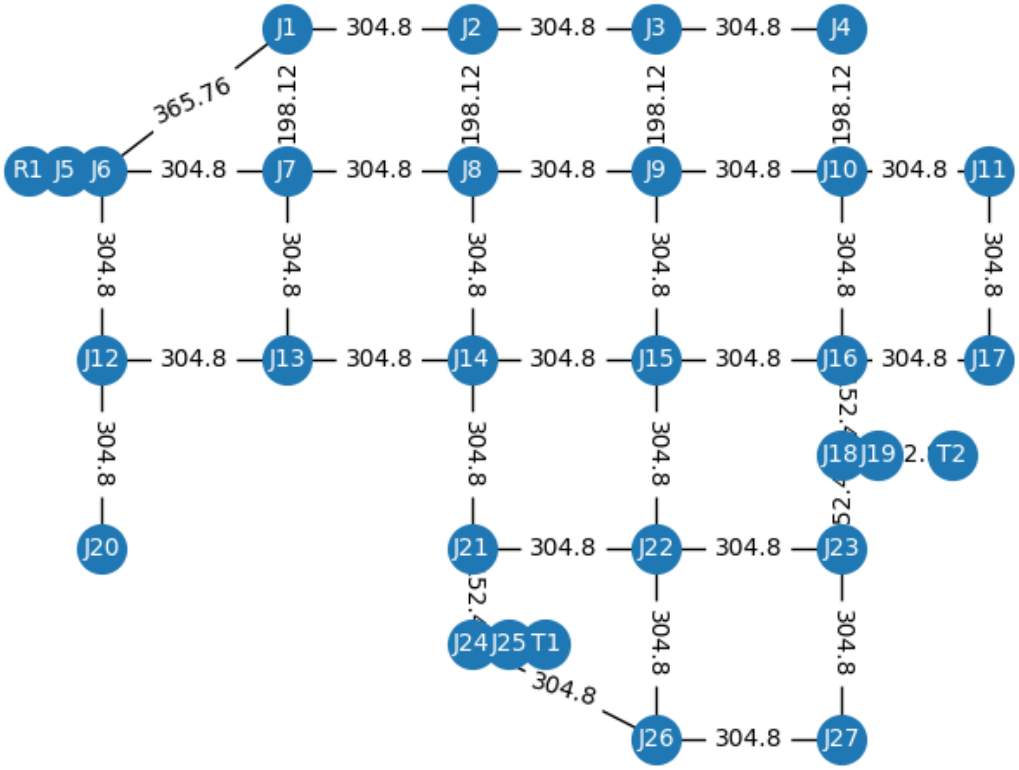


Rys. 7.14 Rezultat wywołania metody *create_scenario* [opracowanie własne]

W pierwszej wersji algorytmu po utworzeniu scenariusza należy na podstawie topologii sieci wodociągowej utworzyć graf nieskierowany z wagami reprezentującymi długość przewodów (w metrach). Na podstawie tego typu grafu (o nieujemnych wagach) wykorzystując Algorytm Dijkstry możliwe będzie określenie najkrótszej ścieżki od ekipy naprawczej do awarii lub magazynu. Prędkość poruszania się ekip naprawczych po mapie (w metrach na godzinę) należy ustawić w pliku konfiguracyjnym symulacji (*config.py*). W celu utworzenia grafu należy wywołać metodę *graph_with_weights*, której przykład przedstawiono poniżej:

```
# ========== Wywołanie metody tworzącej graf nieskierowany
# ========== z wagami:
new_case.graph_with_weights()
```

Na rys. 7.15 przedstawiono rezultat wywołania metody *graph_with_weights* w postaci graficznej.



Rys. 7.15 Rezultat wywołania metody *graph_with_weights* [opracowanie własne]

Proces szeregowania zadań dla ekip naprawczych na podstawie powyższych danych przedstawiono poniżej:

(1) Pobierz awarie z analizowanego poziomu (np. L1).

(2) Ustaw dostępność wszystkich ekip na „dostępny”.

(3) Jeśli zbiór awarii do przeanalizowania jest pusty – zakończ.

(4) Dla każdej ekipy naprawczej wyznacz listę zawierającą zbiór najkrótszych tras od aktualnej lokalizacji ekipy do miejsca wystąpienia poszczególnych awarii.

(5) Na podstawie przewidywanego kosztu naprawy (zużycia materiałów) z list wygenerowanych w punkcie (3) usuń te awarie, które ze względu na braki niezbędnych narzędzi w samochodzie nie mogą zostać naprawione.

(6) Scal i posortuj rosnąco listy otrzymane w (3) punkcie kroku algorytmu (elementem w liście jest: identyfikator ekipy, identyfikator awarii oraz czas dojazdu). Jeśli sumaryczna lista jest pusta, dla każdej ekipy wykonaj punkt (10).

(7) Spośród dostępnych ekip wyznacz ekipę o najmniejszym prawdopodobnym czasie niedostępności.

(8) Dla ekipy posiadającej najmniejszy czas pracy przydziel pierwszą awarię z listy przydzieloną do wskazanego zespołu (na podstawie porównania identyfikatorów ekipy). Następnie usuń awarię z listy i zablokuj ekipę na prawdopodobny czas naprawy i dojazdu na miejsce (miejsce awarii traktowane jest jako węzeł początkowy uszkodzonego przewodu) – przejdź do kroku (9). Jeśli identyfikator ekipy nie znajduje się na liście – przejdź do kroku (10).

(9) Zaktualizuj pozycję ekipy naprawczej, pobierz identyfikator kolejnej ekipy wracając do punktu (7). Jeśli wszystkie ekipy zostały sprawdzone zaktualizuj dane o dojazdach wracając do punktu (3).

(10) Sprawdź zapotrzebowanie sprzętowe i wyznacz trasę do najbliższego magazynu. (zablokuj ekipę na czas dojazdu i pobrania materiałów).

(11) Sprawdź listę awarii do zaszeregowania i na podstawie informacji dobierz potrzebny sprzęt z magazynu – wróć do punktu (3).

Powyższy algorytm wykonywany jest analogicznie dla każdego poziomu zleceń, zakładając, że dane końcowe pierwszej iteracji są początkowymi kolejnej (dane o lokalizacji ekip naprawczych i posiadanych materiałach naprawczych, itp.). W celu wykonania powyższego algorytmu należy wywołać metodę *task_scheduling_ver_a*, przyjmującej jako argument listę poziomów do przeanalizowania:

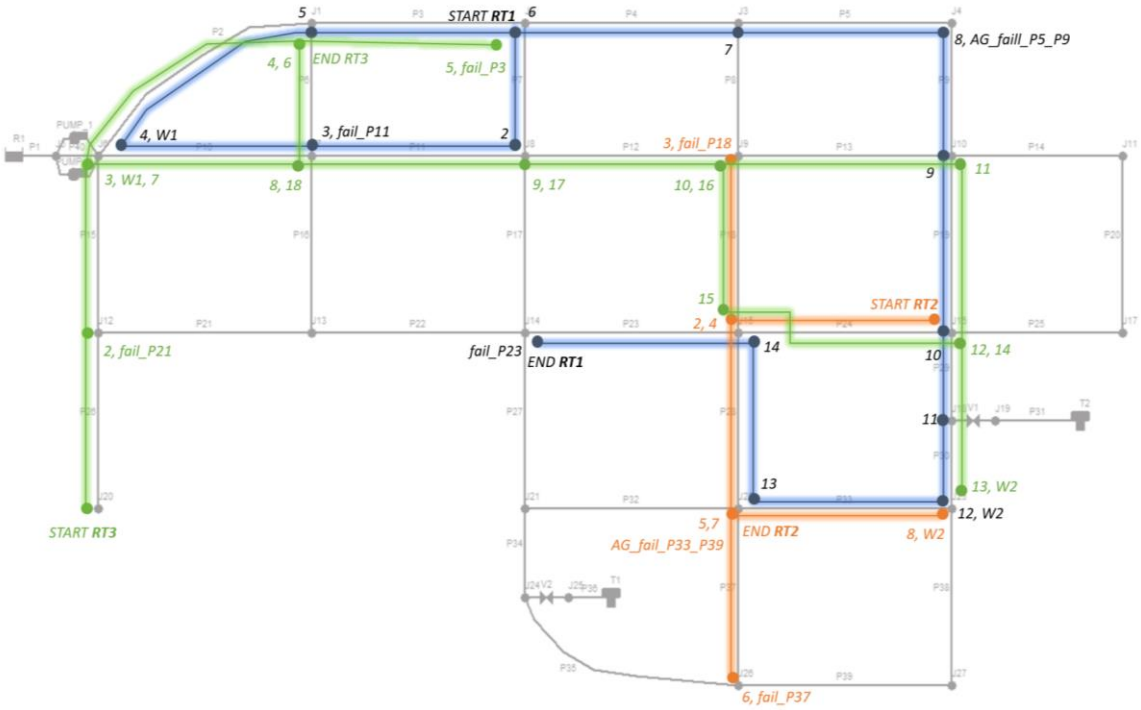
```
# =========== Wywołanie metody task_scheduling_ver_a:
new_case.task_scheduling_ver_a(['L1', 'L2', 'L3', 'L4'])
```

Wynikiem działania *Algorytmu 7* w wersji pierwszej jest zbiór zleceń dla poszczególnej ekipy naprawczej, marszruta obrazująca przemieszczanie się ekip remontowych, informacja o zużytym sprzęcie oraz wskazany czas, w których ekipa

powinna udać się do magazynu. Dla analizowanego przykładu algorytm szeregowania zadań w wersji pierwszej zwrócił następujące wyniki:

- RT1: *fail_P11*, *W1*, *AG_fail_P5_P9*, *W2*, *fail_P23*;
- RT2: *fail_P18*, *fail_P37*, *W2*, *AG_fail_P33_P39*;
- RT3: *fail_P21*, *W1*, *fail_P3*, *W1*, *AG_fail_P14_P20*, *W2*, *fail_P2*.

Całkowity prawdopodobny czas pracy[8] dla poszczególnych ekip przedstawia się następująco (sumaryczny prawdopodobny czas naprawy wszystkich T_{PN} awarii to: 6 dni 10 godzin i 39 minut): RT1 (2 dni, 6 godzin i 16 minut), RT2 (2 dni, 7 godziny i 4 minuty) oraz RT3 (2 dni, 7 godzin i 21 minuty). Na rys. 7.16 przedstawiono graficzną reprezentację tras poszczególnych ekip naprawczych natomiast w Tabeli 7.7 w postaci wykresu Gantta (w celu poprawy czytelności wykresu zaokrąglono czasy do 30 minut).



Rys. 7.16 Trasy poszczególnych ekip naprawczych (Algorytm 7) [opracowanie własne]

[8] Należy pamiętać, że przedstawione czasy prezentują sytuację, w której wykonywane są wszystkie czynności naprawcze i porządkowe. W sytuacji kryzysowej gdyby zrezygnować z szeroko rozumianych czynności porządkowych powyższe czasy uległy by skróceniu o przeszło 20 – 25 procent.

Tab. 7.7 *Wykres Gantta dla analizowanego przykładu działania Algorytmu 7A* [opracowanie własne]

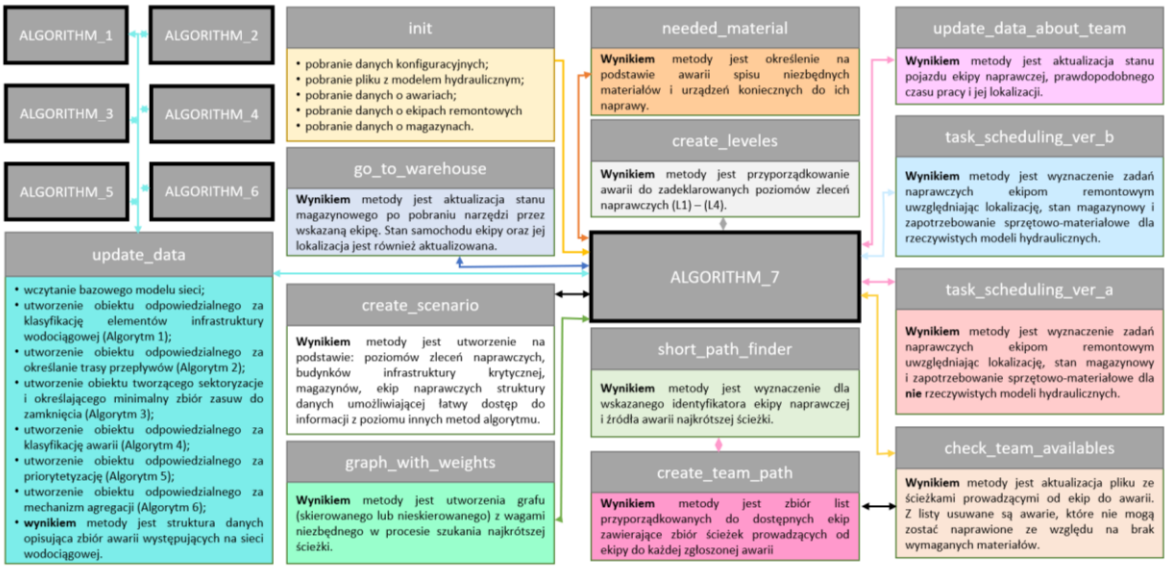
Trasy poszczególnych ekip naprawczych – wykres Gantta

PORA DNIA		RT1	RT2	RT3
	00:00			
	00:30			
	01:00			
	01:30			
	02:00			
	02:30			
	03:00			
	03:30			
	04:00			
	04:30			
	05:00			
	05:30			fail_P21
	06:00			
	06:30			
	07:00			
	07:30			
	08:00		fail_P18	
	08:30			
	09:00			
	09:30			
	10:00			
	10:30			
	11:00			W1
DZIEŃ 1	11:30	fail_P11		
	12:00			DOJAZD
	12:30			
	13:00			
	13:30			
	14:00			
	14:30			
	15:00			
	15:30			
	16:00			
	16:30			
	17:00		DOJAZD	
	17:30			fail_P3
	18:00			
	18:30			
	19:00			
	19:30			
	20:00			
	20:30			
	21:00			
	21:30			
	22:00			
	22:30			
	23:00		fail_P37	W1
	23:30			
	00:00	W1		DOJAZD
	00:30			
	01:00	DOJAZD		
	01:30			
DZIEŃ 2	02:00			
	02:30			
	03:00	AG_fail_P5_P9		
	03:30			
	04:00			
	04:30			

PORA DNIA		RT1	RT2	RT3
	05:00			
	05:30		W2	
	06:00			
	06:30		DOJAZD	
	07:00			
	07:30			
	08:00			
	08:30			
	09:00			
	09:30			
	10:00			
	10:30			
	11:00			
	11:30			
	12:00			
	12:30	AG_fail_P5_P9		
	13:00			fail_P14_P20
	13:30			
DZIEŃ 2	14:00			
	14:30			
	15:00			
	15:30			
	16:00			
	16:30			
	17:00			
	17:30			
	18:00			
	18:30			
	19:00		AG_fail_P33_P39	
	19:30			
	20:00			
	20:30			
	21:00	W2		
	21:30			
	22:00	DOJAZD		W2
	22:30			
	23:00			DOJAZD
	23:30			
	00:00			
	00:30			
	01:00			
	01:30			
	02:00			
	02:30	fail_P23		
	03:00			fail_P2
	03:30			
	04:00			
	04:30			
DZIEŃ 3	05:00			
	05:30			
	06:00			
	07:00			
	07:30			
	08:00			
	08:30			
	09:00			
	09:30			
	10:00			

Sposób działania algorytmu w wersji drugiej jest identyczny, różnią się natomiast dane wejściowe. W przypadku rzeczywistych modeli w plikach dotyczących awarii, ekip naprawczych, czy magazynów w atrybutach dotyczących lokalizacji umieszczane są rzeczywiste współrzędne geograficzne (w systemie odniesienia WGS-84). Następnie na podstawie ogólnodostępnych baz mapowych (OpenStreetMap) oraz biblioteki OSMnx[9] pobierany jest graf reprezentujący drogi publiczne (w tym przypadku jako graf skierowany). Następnie w oparciu o dane o awariach i mapie drogowej algorytm szeregowania wyznacza zlecenia naprawcze poszczególnym ekipom. Przykład scenariusza testowego dla rzeczywistego modelu sieci wodociągowej Środy Wielkopolskiej został przedstawiony w Rozdziale 8.

Na rys. 7.17 przedstawiono schemat przepływów danych między zaimplementowanymi metodami w opracowanym algorytmie, natomiast w *Załączniku M* niniejszej rozprawy zawarto kod algorytmu szeregowania zadań (*Algorytm 7*).



Rys. 7.17 Schemat przepływu danych *Algorytmu 7* [opracowanie własne]

[9] Więcej o bibliotece OSMnx można znaleźć na: https://osmnx.readthedocs.io/en/stable/index.html

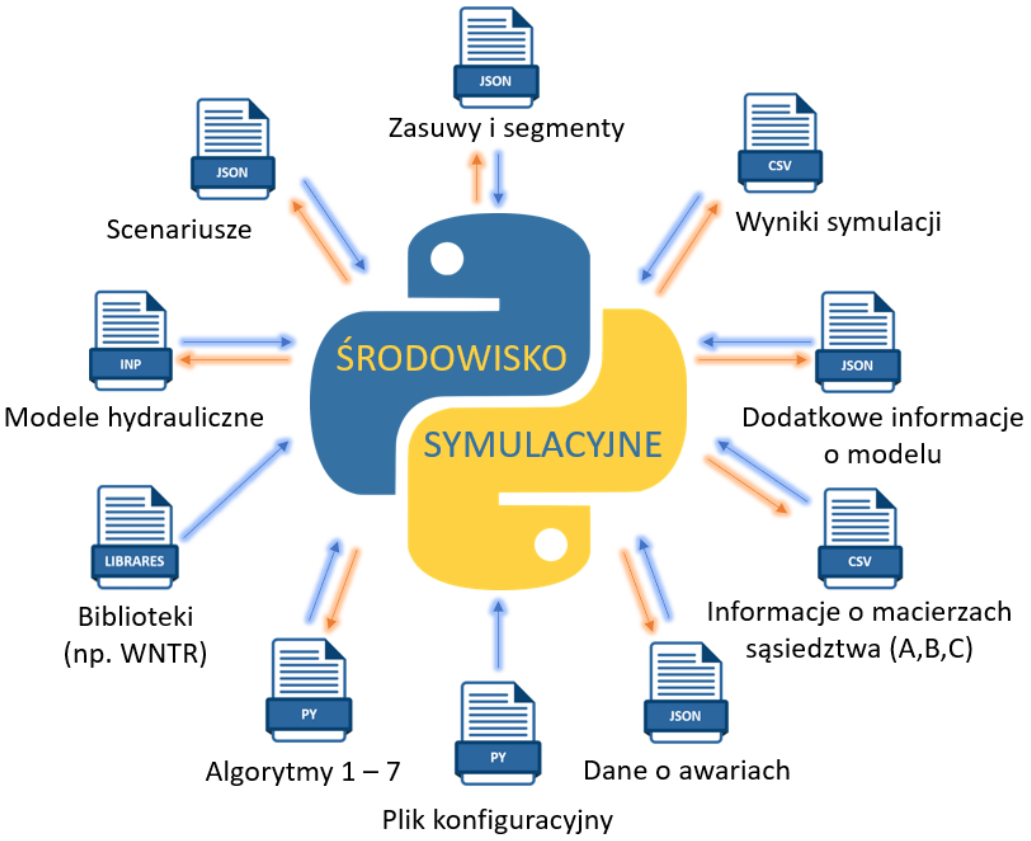
8. System Wspomagania Decyzji w procesie przywracania ciągłości dystrybucji wody po wystąpieniu sytuacji katastroficznej

8.1 Wprowadzenie

W niniejszym rozdziale przedstawiono wyniki badań symulacyjnych dotyczących zaproponowanej metodyki poprawy efektywności procesu przywracania ciągłości dystrybucji wody w sytuacji pokatastroficznej. Wyniki opracowano na podstawie analizy pracy modelu nierzeczywistego systemu dystrybucji wody oraz modelu opisującego pracę rzeczywistego modelu sieci wodociągowej w mieście Środa Wielkopolska. Model hydrauliczny rzeczywistego systemu zbiorowego zaopatrzenia w wodę został udostępniony przez *Miejskie Przedsiębiorstwo Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej*. Przyjmuje się, że model ten jest poprawnie zwalidowany i uzyskane w nim wyniki są wiarygodne. Dla każdego modelu przygotowano zestaw dwóch losowo wygenerowanych scenariuszy testowych różniących się: liczbą i typem awarii oraz liczbą dostępnych ekip naprawczych i magazynów.

8.2 Środowisko symulacyjne

Modele hydrauliczne analizowanych sieci wodociągowych zostały przygotowane z wykorzystaniem środowiska EPANET, opracowanego i aktualnie rozwijanego przez Agencję Ochrony Środowiska USA. Oprogramowanie udostępniane jest na zasadach licencji publicznej, umożliwiającej wykorzystanie zarówno samego programu komputerowego jak i jego kodów źródłowych. W celu zautomatyzowania czynności klasyfikacyjnych i analizujących działanie modeli hydraulicznych w różnych warunkach wykorzystano język skryptowy Python wraz z biblioteką WNTR, które zostały szczegółowo omówione w Rozdziale 4 niniejszej rozprawy. Obliczenia wykonano na jednostce komputerowej charakteryzującej się następującymi parametrami: procesor Intel® Core™ i7-7700HQ CPU @ 2.80GHz, pamięcią RAM 16 GB oraz dyskiem SSD 1 TB. Na komputerze zainstalowany został system operacyjny Microsoft Windows 10 w wersji Home. Utworzone przez autora pracy skrypty w języku Python realizują algorytmy przedstawione w Rozdziale 7. Koncepcję integracji wykorzystanych narzędzi tworzących środowisko symulacyjne przedstawiono na rys. 8.1.



Rys. 8.1 Schemat integracji narzędzi tworzących środowisko symulacyjne [opracowanie własne]

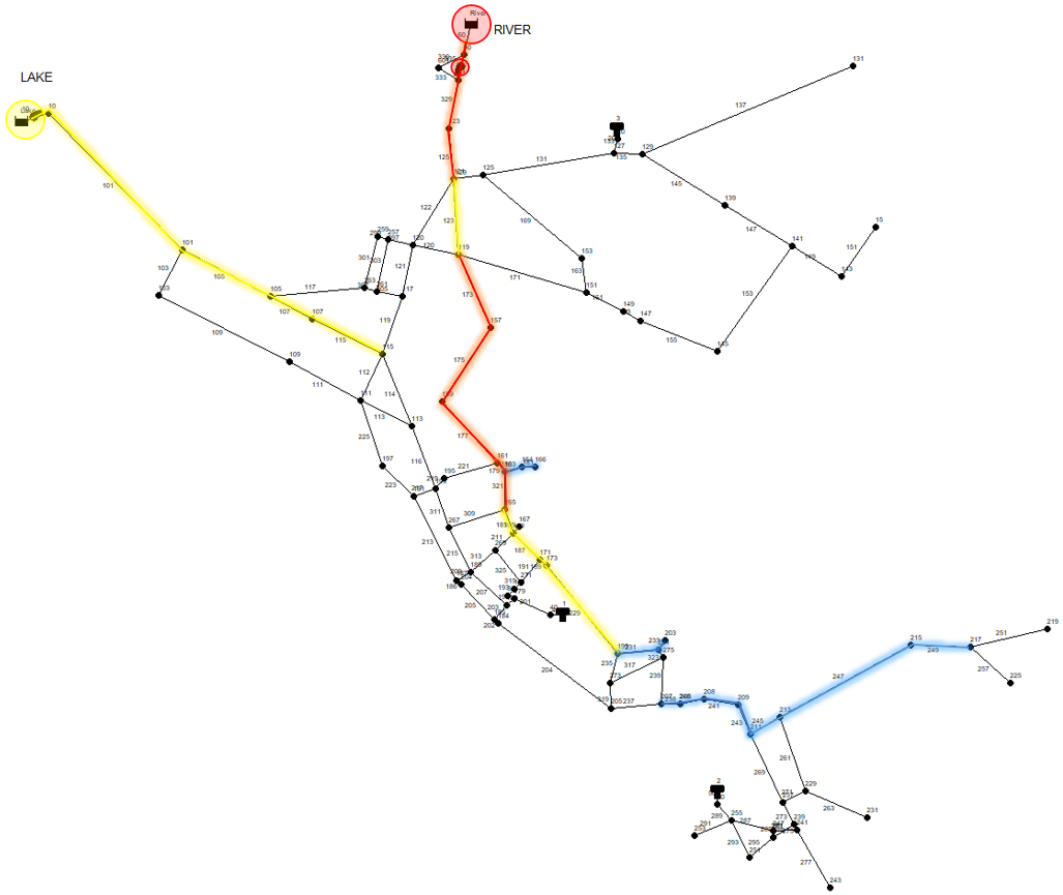
8.3 Charakterystyka wybranych modeli hydraulicznych

W celu analizy wyników i poprawności działania zaproponowanych algorytmów zdecydowano się na przetestowaniu ich na dwóch różnych modelach hydraulicznych pod względem liczby węzłów, przewodów i struktury topologii. W przypadku pierwszego modelu (nierzeczywistego) ze względu na brak informacji dotyczących zasuw wygenerowano je w sposób losowy zakładając, że liczba zasuw nie może być większa niż 40 procent liczby dostępnych przewodów. W przypadku modelu rzeczywistego pomimo wiedzy o stanie i lokalizacji zasuw ze względów umownych (dotyczących przekazania modelu, danych wrażliwych i bezpieczeństwa) zdecydowano się na ten sam krok. W trakcie budowy modeli hydraulicznych uwzględniających lokalizację zasuw i przygotowywania scenariuszy symulacyjnych założono, że każda referencyjna (bazowa) symulacja hydrauliczna obrazująca pracę sieci wodociągowej w warunkach

normalnych trwać będzie 24 godziny (z krokiem jedna godzina) oraz dotyczyć będzie dnia o najwyższym zapotrzebowaniu w roku.

8.3.1 Model hydrauliczny 1 – Net3

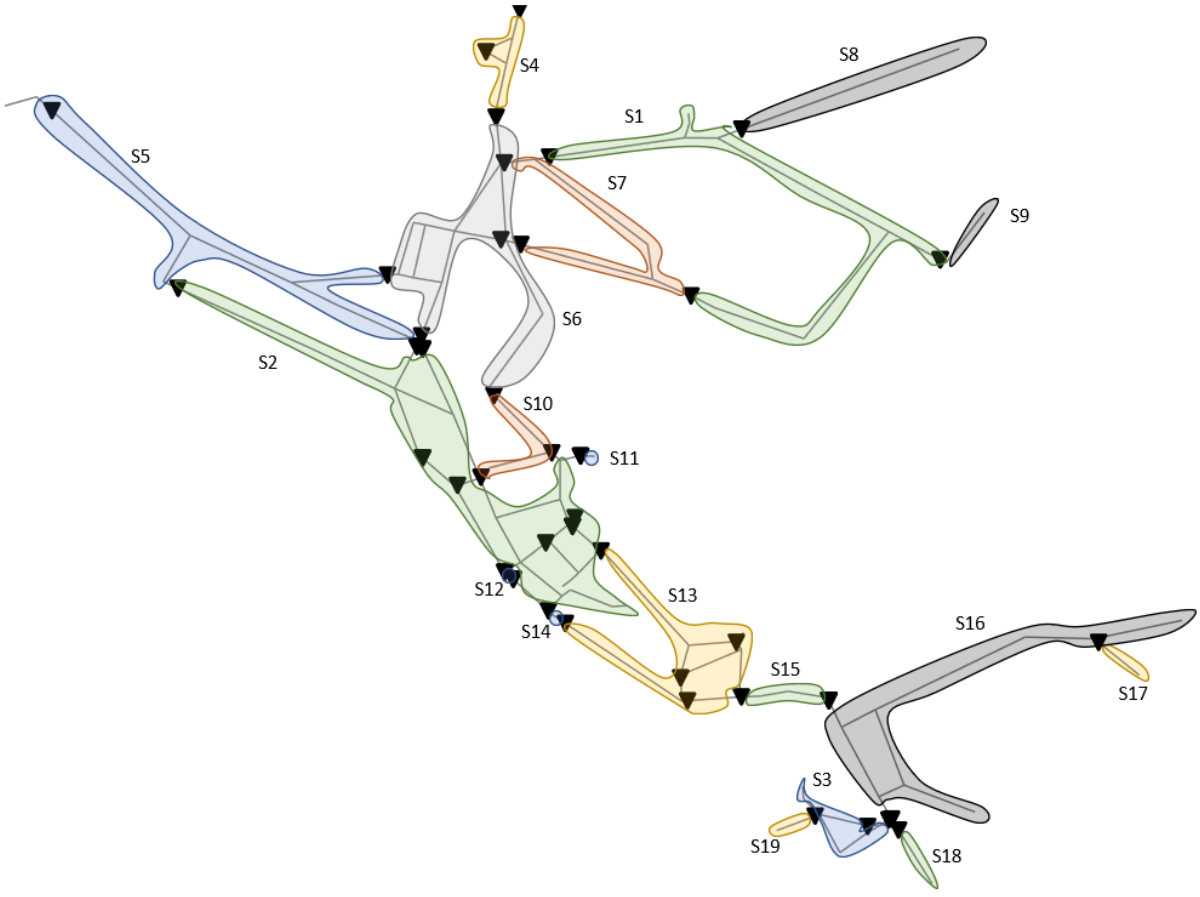
Pierwszym analizowanym modelem hydraulicznym jest przykładowy model sieci zaprojektowany przez twórców środowiska programistycznego i silnika obliczeń EPANET. Sieć nosi nazwę Net3 i zawiera: 92 węzły, 2 źródła wody (rezerwuary), 3 zbiorniki, 117 przewodów oraz 2 pompy. Sieć ma charakter sieci mieszanej, w większości zaprojektowana w systemie obwodowym z niewielką liczbą rozgałęzień. Na rys. 8.2 przedstawiono topologię sieci o nazwie Net3.



Rys. 8.2 Topologia sieci [opracowanie własne]

Na rys. 8.2 przedstawiono również rezultat działania *Algorytmu 1* wyznaczającego krytyczność poszczególnych elementów infrastruktury wodociągowej. Na rysunku

przedstawiono wyłącznie elementy katgorii CAT-2, CAT-3 oraz CAT-4. Kolorem niebieskim oznaczono elementy o niewielkim wpływie na proces dystrybucji (CAT-2). Do zbioru tych elementów zaliczamy przewody o identyfikatorach: 180, 181, 231, 233, 238, 241, 243, 245, 247 oraz 249. Kolorem żółtym oznaczono wysoką krtytyczność danego elementu (CAT-3). Do elementów powyższej kategorii zaliczamy przewody o identyfikatorach 123, 101, 105, 107, 115, 123, 183, 187, 189, 229, pompę o identyfikatorze 10 oraz źródło (rezerwuar) o identyfikatorze *Lake*. Elmenty krytyczne, bez których sieć nie może funkcjonować oznaczone kolorem czerwonym (CAT-4) to źródło wody (rezerwuar) o identyfikatorze *River*, pompa o identyfikatorze 335 oraz natępujace przewody: 60, 329, 125, 173, 175, 177, 170 oraz 321. Pozostałe elementy zostały sklasyfikowane do grupy CAT-1. Dla przykładowego modelu wygenerowano w sposób losowy zestaw 51 zasuw.

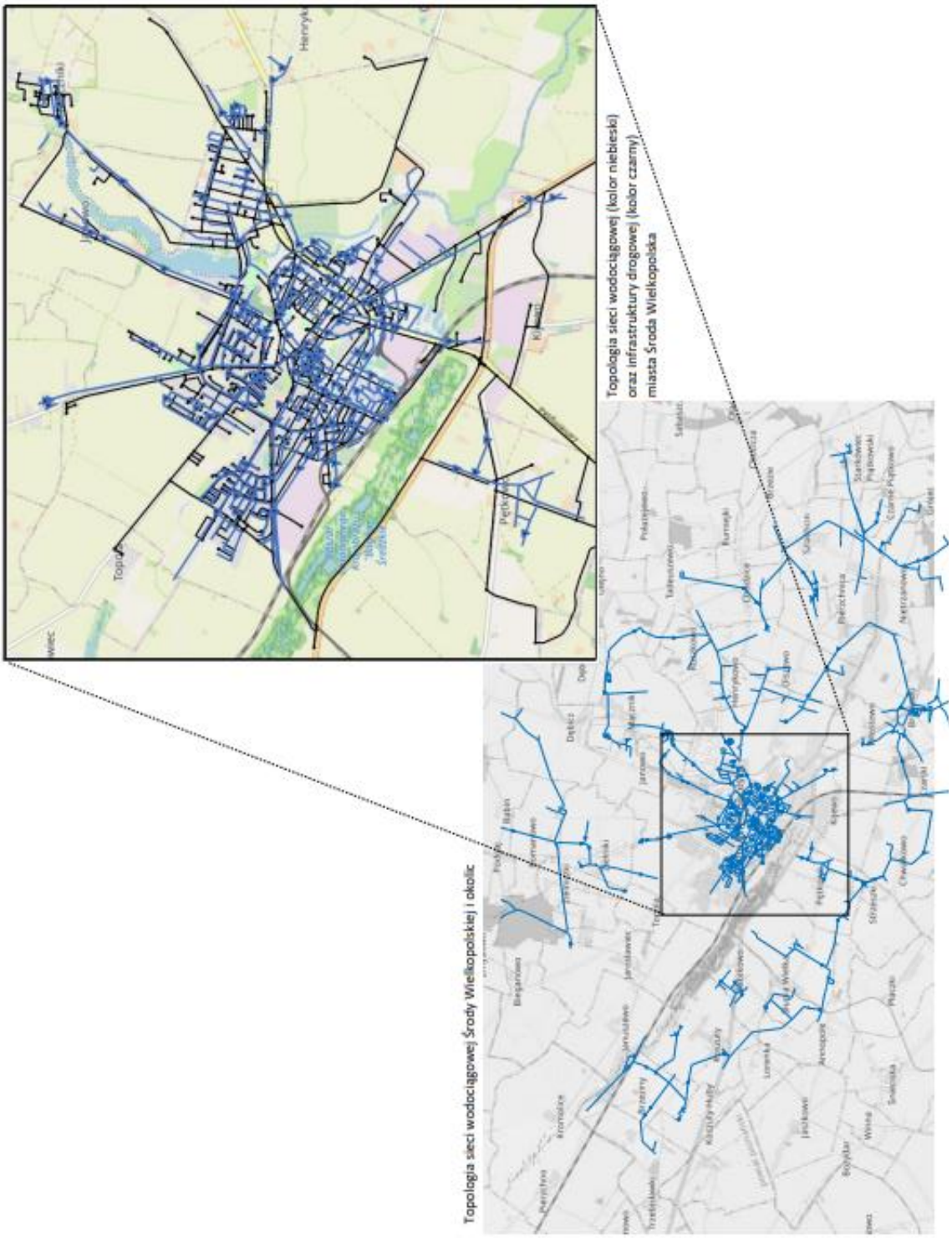


Rys. 8.3 Topologia sieci Net3 z lokalizacją zasuw i sektorów [opracowanie własne]

Na rys. 8.3 przedstawiono lokalizację zasuw oraz wynik działania *Algorytmu 3* odpowiedzialnego za utworzenie bazy wiedzy na temat istniejących w sieci segmentów. Algorytm podzielił sieć na 19 segmentów. Okazuje się, że losowe rozmieszczenie zasuw skutkuje wystąpieniem sytuacji, w których zasuwa ze względu na swoją lokalizację nie spełnia swojego zadania. Przypadek ten można zauważyć np. w segmencie (S2), który składa się aż z 26 zasuw (mimo, że do odizolowania tego segmentu wystarczy zamknąć 6). Tego typu sytuacja pozwoli zweryfikować działanie algorytmu minimalizacji liczby zasuw do zamknięcia wykorzystujący ślad wody (*Algorytm 2*).

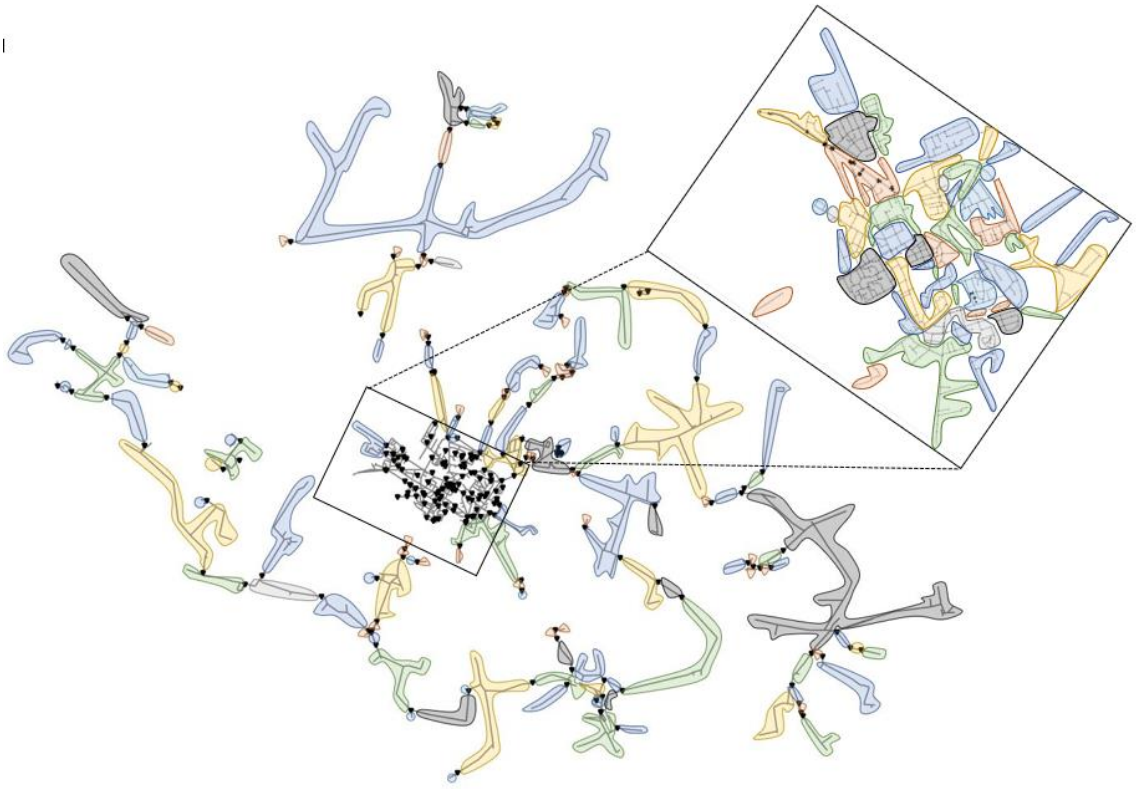
8.3.2 Model hydrauliczny 3 – Środa Wielkopolska

Model hydrauliczny rzeczywistego systemu zbiorowego zaopatrzenia w wodę symuluje pracę sieci wodociągowej Środy Wielkopolskiej. Miasto o powierzchni 17.98 km^2 zlokalizowane jest w województwie wielkopolskim. Sieć dostarcza wodę do ponad 23 tys. odbiorców, a jej model został udostępniony przez *Miejskie Przedsiębiorstwo Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej*. Sieć wodociągowa składa się z: 1400 węzłów, 7 źródeł wody (rezerwuary), 1523 przewodów oraz 35 zasuw (typu TCV i PRV). Na rys. 8.4 przedstawiono topologię sieci wodociągowej miasta Środa Wielkopolska i okolic oraz topologię sieci wodociągowej (kolor niebieski) wraz z infrastrukturą drogową (kolor czarny) miasta.



Rys. 8.4 Topologia sieci Środy Wielkopolskiej [opracowanie własne]

Ze względów umownych między autorem pracy, a *Miejskim Przedsiębiorstwem Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej* wyniki działania algorytmu wskazującego elementy krytyczne sieci wodociągowej (*Algorytm 1*) nie zostaną w pracy przedstawione. Wyniki te natomiast zostaną wykorzystane w procesie decyzyjnym szeregowania zadań ekipom naprawczym. Na rys. 8.5 przedstawiono natomiast lokalizację zasuw oraz wynik działania *Algorytmu 3* odpowiedzialnego za utworzenie bazy wiedzy na temat istniejących w sieci segmentów dla całego systemu zbiorowego zaopatrzenia w wodę (miasto i okolice).



Rys. 8.5 Topologia sieci Środy Wielkopolskiej z lokalizacją zasuw i sektorów [opracowanie własne]

8.4 Przykładowe scenariusze zdarzeń katastroficznych (dane wejściowe)

Dla każdego z modeli opisanych w Rozdziale 8.3 przygotowano zestaw dwóch scenariuszy testowych (bez i z uwzględnieniem procesu agregacji), celem których jest sprawdzenie zaproponowanej metodyki. Celem powyższego zabiegu jest sprawdzenie

wpływu procesu agregacji na całkowity prawdopodobny czas naprawy. W Tabeli 8.1 przedstawiono charakterystykę poszczególnych scenariuszy.

Tab. 8.1 *Informacje o scenariuszach testowych [opracowanie własne]*

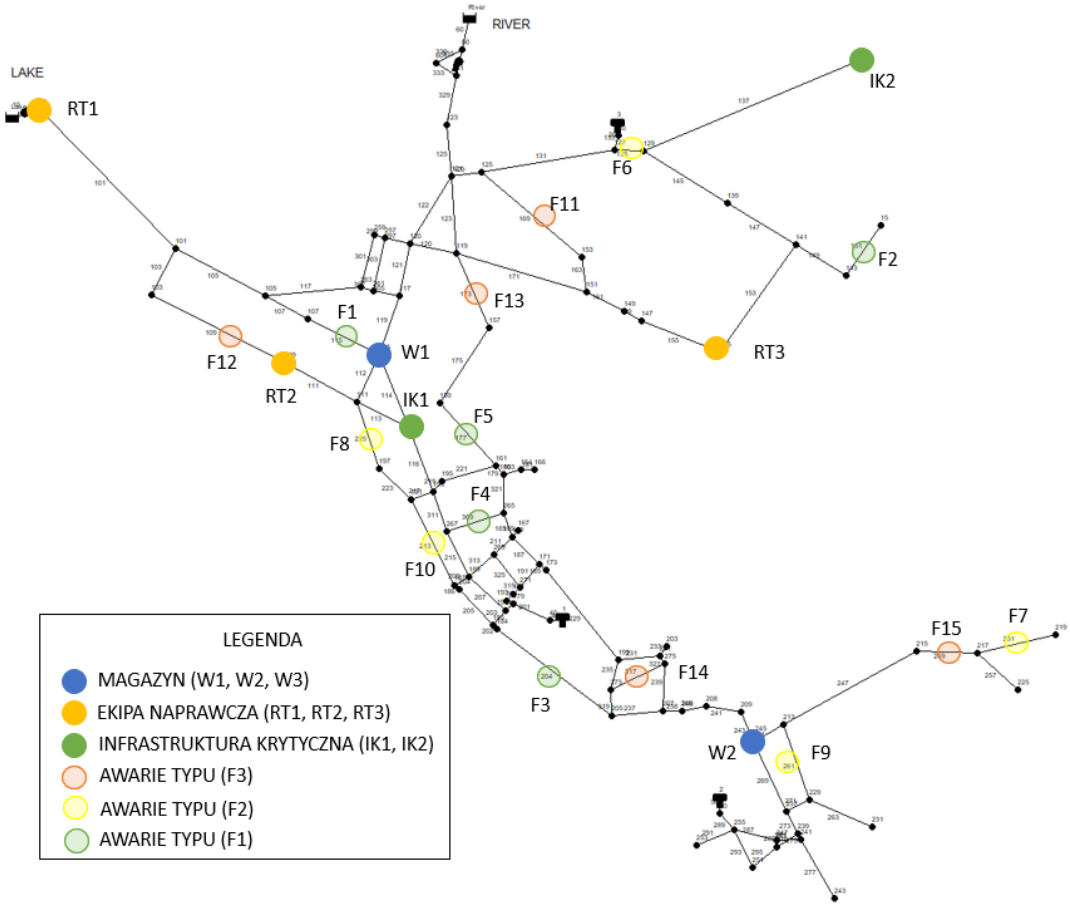
Informacje o scenariuszach testowych				
ID SIECI	**Net3**		**Środa WLKP**	
	Scenario_1	Scenario_2	Scenario_1	Scenario_2
Liczba awarii typu F1	5	3	8	10
Liczba awarii typu F2	5	7	10	15
Liczba awarii typu F3	5	8	10	12
Liczba ekip remontowych	3	3	4	4
Liczba magazynów	2	3	3	4
Liczba budynków krytycznych	2	2	4	2

W trakcie opracowywania scenariuszy testowych założono, że każda z ekip maksymalnie będzie mogła przewieźć: 15 zacisków (*clamps*), 12 przewodów (*pipes*) oraz 45 narzędzi i materiałów innego typu (*other stuff*). Lokalizacja magazynów oraz ekip naprawczych będzie ustalana w sposób losowy.

8.4.1 Scenariusze testowe – model Net3

8.4.1.1 Scenariusz pierwszy (Scenario_1)

Pierwszym analizowanym scenariuszem testowym (Scenario_1) jest przypadek wystąpienia trzęsienia ziemi w miejscowości reprezentowanej za pośrednictwem modelu Net3. Sumaryczna liczba awarii wyniosła piętnaście (po pięć z każdego typu). Dostępne są trzy ekipy naprawcze, dwa magazyny z niezbędnymi narzędziami i materiałami oraz dwa budynki użyteczności publicznej, które zaliczone zostały do infrastruktury krytycznej. Na rys. 8.6 przedstawiono miejsce występowania (w tym typ awarii), lokalizację magazynów, budynków infrastruktury krytycznej oraz ekip naprawczych.

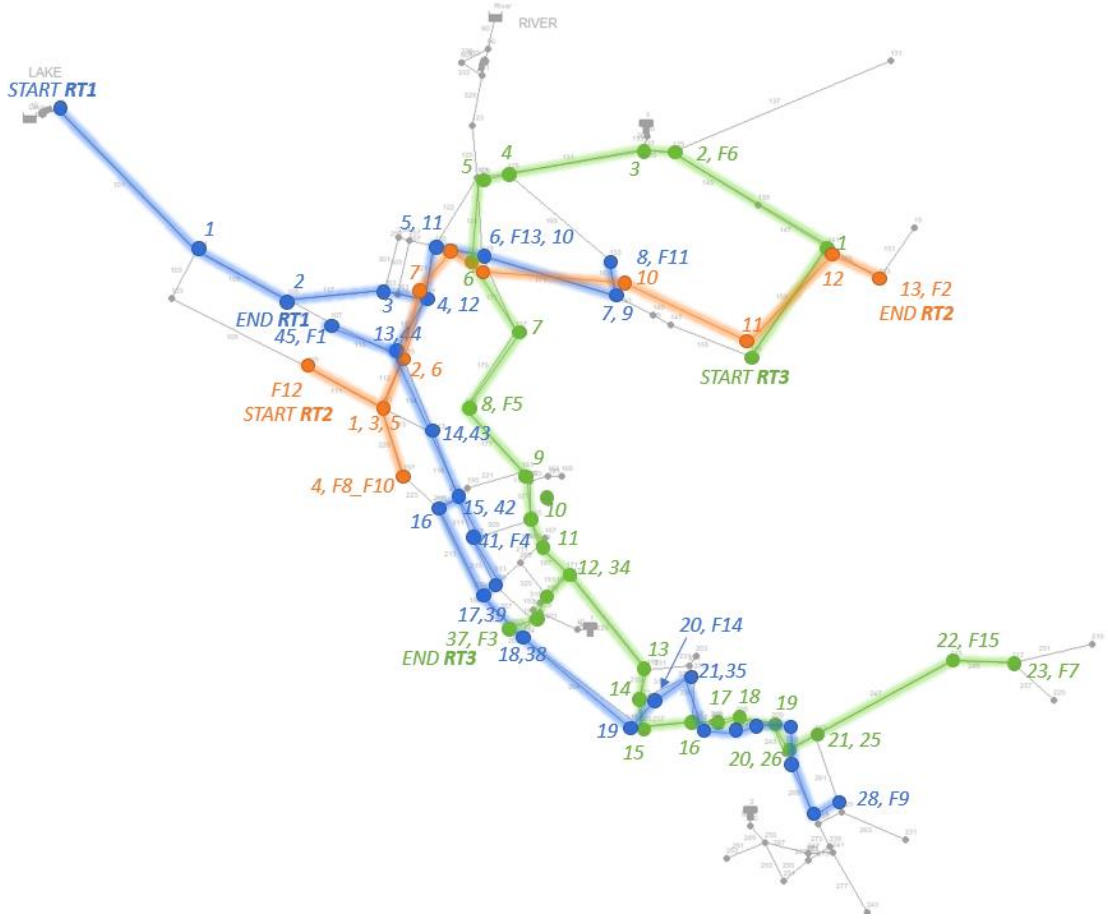


Rys. 8.6 Scenariusz testowy Net3 – Scenario_1 [opracowanie własne]

Na podstawie informacji o rodzaju uszkodzenia sieci wodociągowej *Algorytm 4* dokonał następującej klasyfikacji:

- **kategoria C1**: F5, F8, F10, F12, F13;
- **kategoria C2**: F6, F7, F9, F11, F14, F15;
- **kategoria C3**: F1, F2, F3, F4.

Algorytm priorytetyzacji wskazał awarię o identyfikatorze *F6* jako awarię priorytetową, ponieważ uniemożliwia transport wody do budynku infrastruktury krytycznej *IK2*. *Algorytm 6* zaproponował natomiast agregację awarii *F8* z *F10* w celu zaoszczędzenia sumarycznej liczby zasuw do zamknięcia. Na rys. 8.7 przedstawiono rezultat algorytmu szeregowania zadań z uwzględnieniem agregacji.



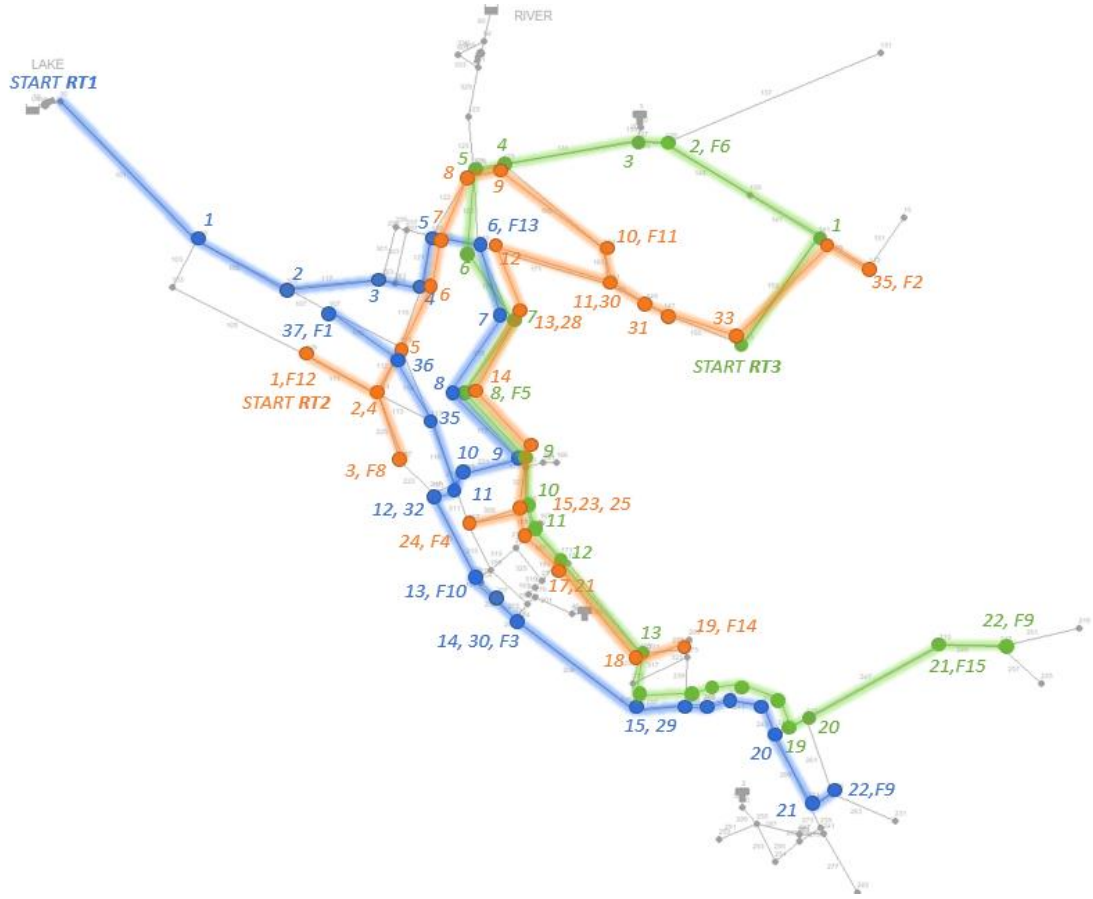
Rys. 8.7 Rozwiązanie problemu szeregowania zadań – Scenario_1 (z agregacją) [opracowanie własne]

Całkowity prawdopodobny czas naprawy wszystkich awarii wyniósłby około 9 dni 11 godzin i 3 minuty. W przypadku *Algorytmu 7* wyznaczającego trasę poszczególnym ekipom naprawczym (z uwzględnieniem agregacji) otrzymano następującą propozycję:

- **RT1**: F13, F11, W1, F14, F9, W2, F4, F1;
- **RT2**: F12, W1, F8_F10, W1, F2;
- **RT3**: F6, F5, W2, F15, F7, F3.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 2 dni, 22 godziny i 22 minuty, RT2 – 2 dni, 23 godziny i 44 minuty oraz RT3 – 3 dni, 5 godzin i 20 minut. Ekipy na dojazd do awarii oraz magazynów potrzebowały

kolejno: 3 godziny 15 minut, 1 godzinę i 42 minuty oraz 3 godziny 27 minut. Na rys. 8.8 przedstawiono rezultat algorytmu szeregowania zadań bez uwzględniania agregacji.



Rys. 8.8 Rozwiązanie problemu szeregowania zadań – Scenario_1 (bez agregacji) [opracowanie własne]

Algorytm wyznaczania tras nieuwzględniającego agregacji zaproponował następujące rozdysponowanie zleceń naprawczych.

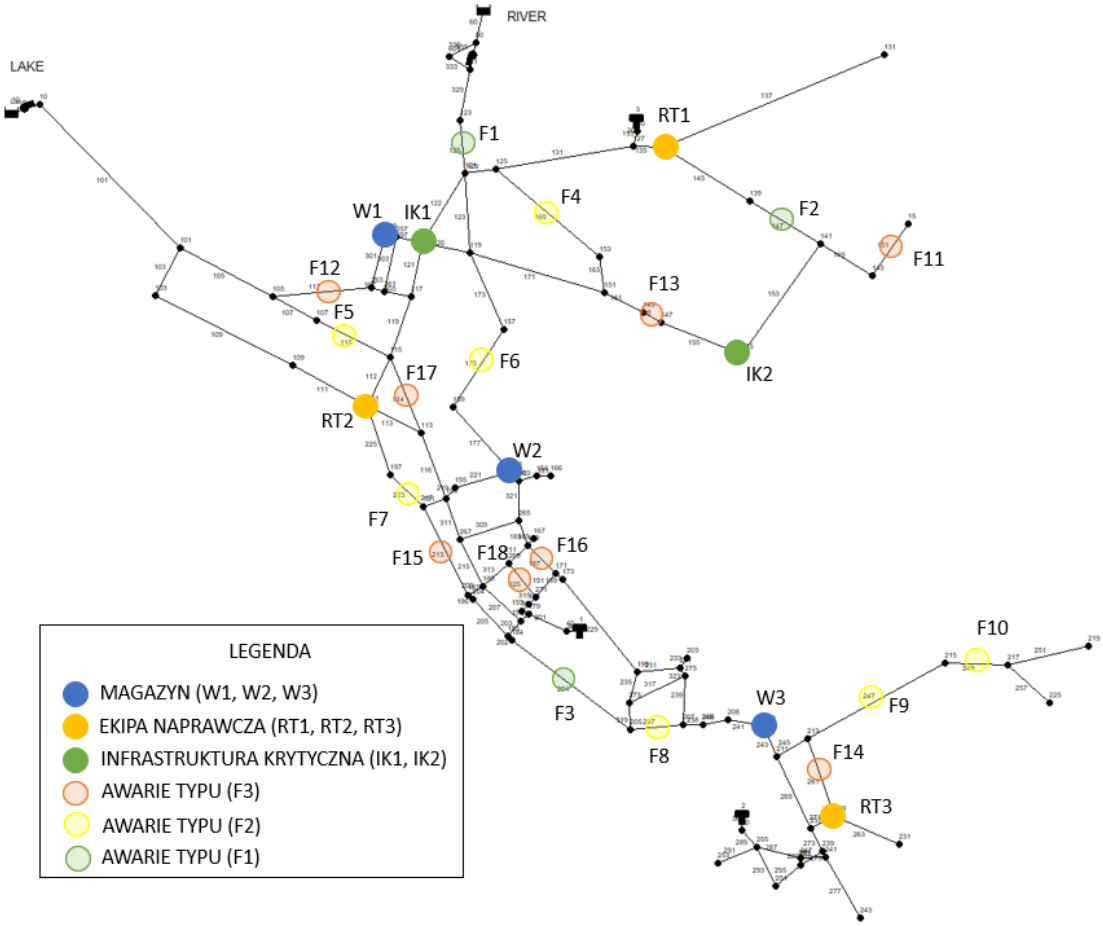
- **RT1**: F13, F10, W2, F9, F3, F1;
- **RT2**: F12, F8, W1, F11, F14, F4, W1, F2;
- **RT3**: F6, F5, W2, F15, F7.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 3 dni, 5 godzin i 28 minut, RT2 – 3 dni, 4 godziny i 9 minut oraz RT3 – 2 dni, 23

godziny i 24 minuty. Ekipy na dojazd do awarii oraz magazynów potrzebowały kolejno: 3 godziny 2 minut, 2 godziny 40 minut oraz 3 godzinę i 8 minuty.

8.4.1.2 Scenariusz drugi (Scenario_2)

Drugim analizowanym scenariuszem testowym (Scenario_2) jest przypadek wystąpienia wielu awarii, w którym sumaryczna liczba awarii wyniosła 18 (3 awarie typu F1, 7 awarii typu F2 oraz 8 awarii typu F3). Analogicznie do przykładu pierwszego liczba dostępnych ekip remontowych wynosi trzy, a liczba budynków infrastruktury krytycznej dwa. W tym przypadku testowym dostępne są trzy magazyny, których lokalizacja została wylosowana. Na rys. 8.9 przedstawiono miejsce występowania awarii, lokalizację magazynów, budynków infrastruktury krytycznej oraz ekip naprawczych.

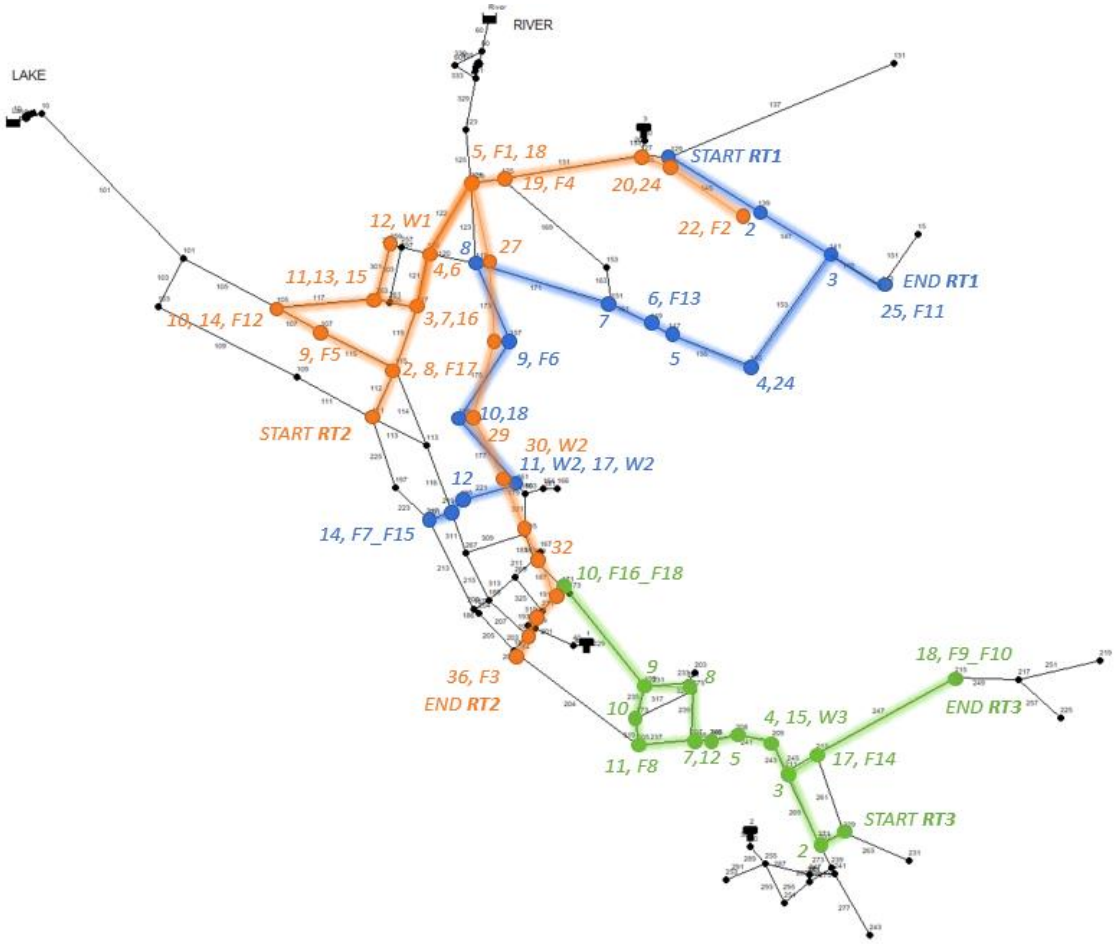


Rys. 8.9 Scenariusz testowy Net3 – Scenario_2 [opracowanie własne]

Na podstawie informacji o rodzaju uszkodzenia sieci wodociągowej *Algorytm 4* dokonał następującej klasyfikacji:

- **kategoria C1**: F1, F6, F7, F15, F16, F17, F18
- **kategoria C2**: F2, F4, F5, F8, F11, F12, F13, F14, F9, F10;
- **kategoria C3**: F3.

Algorytm priorytetyzacji wskazał awarię o identyfikatorze *F13* jako awarię priorytetową, ponieważ uniemożliwia transport wody do budynku infrastruktury krytycznej *IK2*. *Algorytm 6* zaproponował natomiast agregację następujących awarii: *F9* z *F10*, *F7* z *F15* oraz *F16* z *F18*. Na rys. 8.10 przedstawiono rezultat algorytmu szeregowania zadań z agregacją.

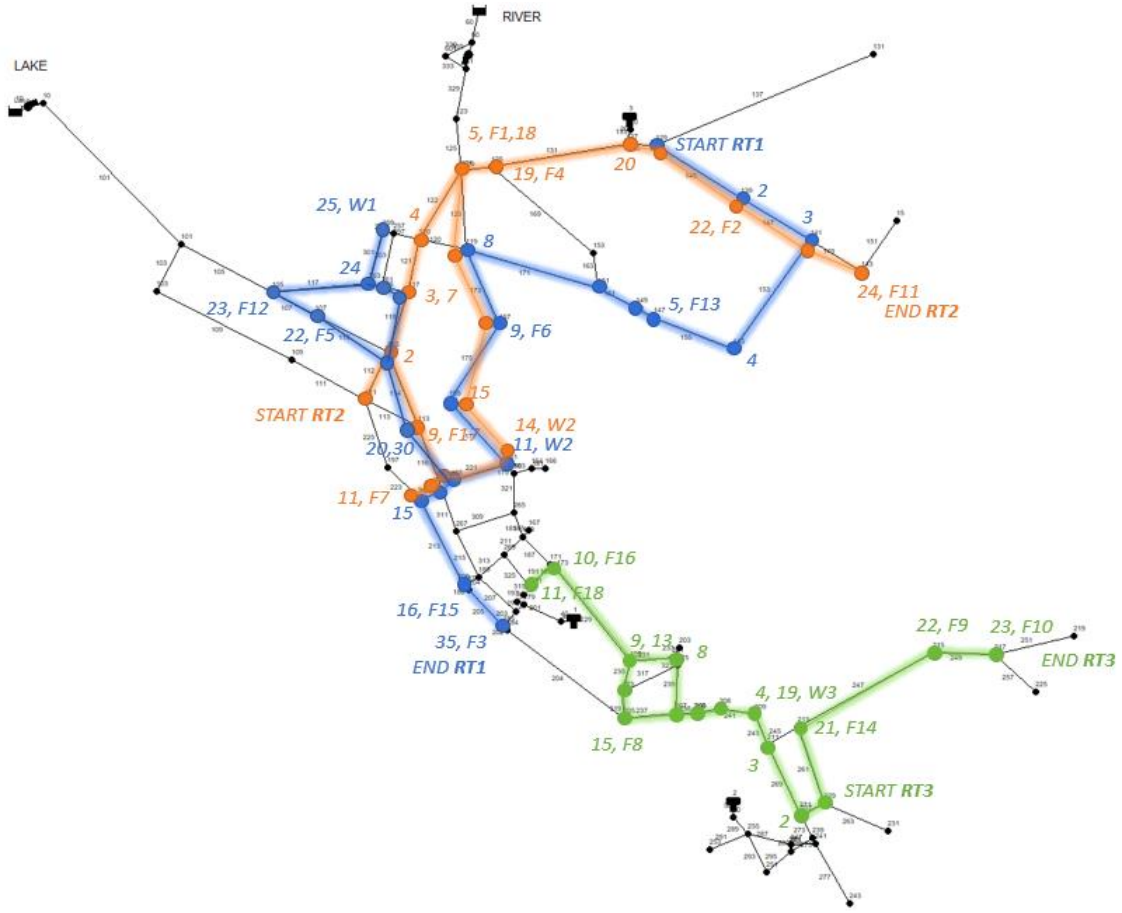


Rys. 8.10 Rozwiązanie problemu szeregowania zadań – Scenario_2 (z agregacją) [opracowanie własne]

Całkowity prawdopodobny czas naprawy wszystkich awarii (nie wliczając dojazdu) wyniósłby około 11 dni 13 godzin i 5 minuty. W przypadku *Algorytmu 7* wyznaczającego trasę poszczególnym ekipom naprawczym (z uwzględnieniem agregacji) otrzymano następującą propozycję:

- **RT1**: F13, F6, W2, F7_F15, W2, F11;
- **RT2**: F1, F17, F5, W1, F12, F4, F2, W2, F3;
- **RT3**: F16_F18, F8, W3, F14, F9_F10.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 3 dni, 19 godzin i 26 minut, RT2 – 3 dni, 23 godzin i 32 minuty oraz RT3 – 4 dni, 6 godziny i 2 minuty. Na rys. 8.11 przedstawiono rezultat algorytmu szeregowania zadań bez uwzględniania agregacji.



Rys. 8.11 Rozwiązanie problemu szeregowania zadań – Scenario_2 (bez agregacji) [opracowanie własne]

Wersja algorytmu wyznaczania tras nie uwzględniającego agregacji zaproponowała następujące rozdysponowanie zleceń naprawczych.

- **RT1**: F13, F6, W2, F15, F5, F12, W1, F3;
- **RT2**: F1, F17, F7, W2, F4, F2, F11;
- **RT3**: F16, F18, F8, W3, F14, F9, F10.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 3 dni, 23 godziny i 17 minut , RT2 – 3 dni, 20 godzin i 57 minut oraz RT3 – 4 dni, 8 godzin i 16 minut.

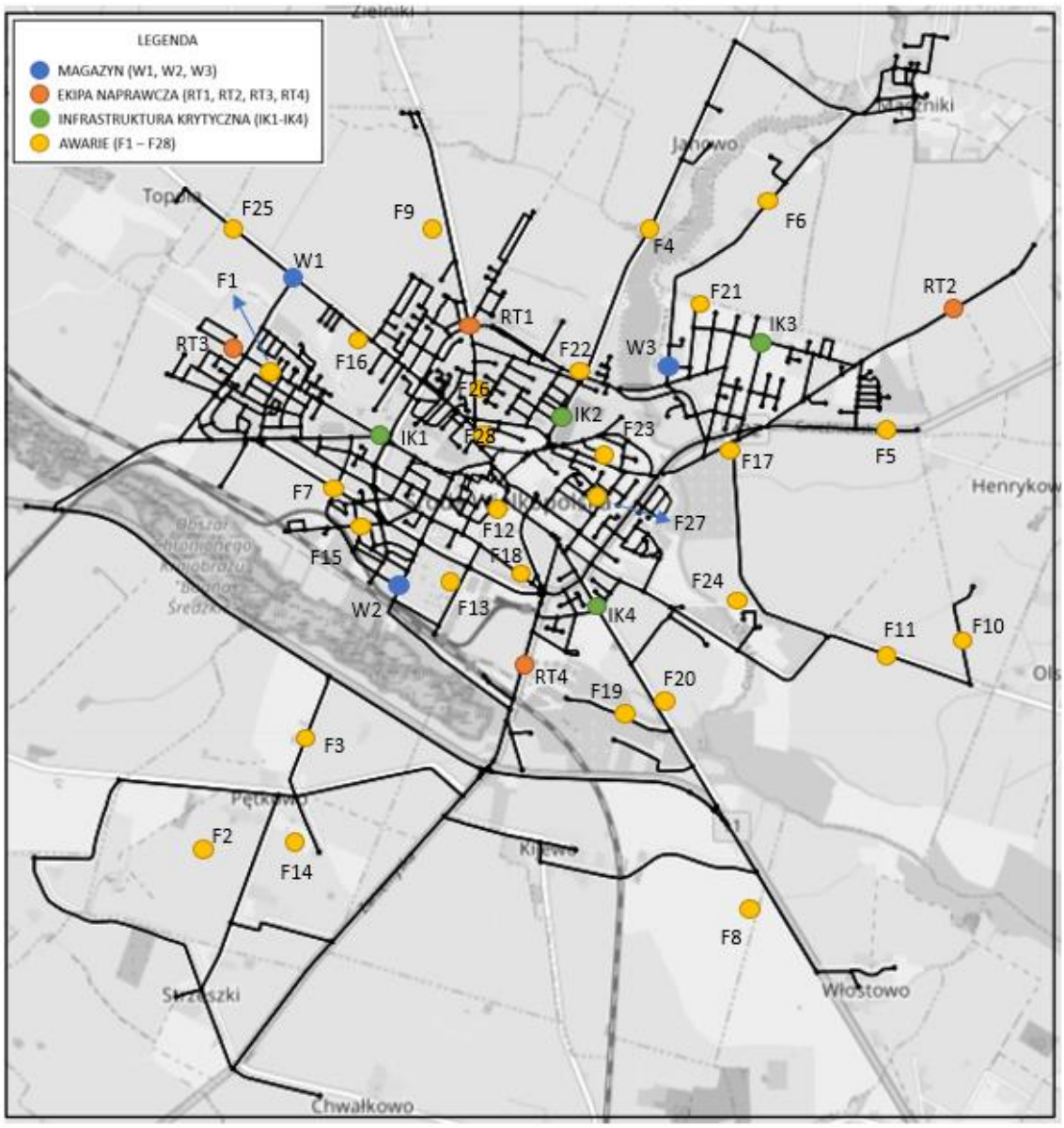
8.4.2 Scenariusze testowe – model Środy Wielkopolskiej

8.4.2.1 Scenariusz pierwszy (Scenario_1)

Pierwszym analizowanym scenariuszem testowym (Scenario_1) dla rzeczywistego modelu miasta Środa Wielkopolska jest przypadek wystąpienia wielu awarii spowodowanych sytuacją katastroficzną. Zidentyfikowano 28 awarii, w tym 8 typu (F1) o identyfikatorach: F1 – F8, 10 typu (F2) o identyfikatorach F9 – F18 oraz 10 typu (F3) o identyfikatorach F19 – F28. Miasto dysponuje czteroma ekipami naprawczymi (RT1 – RT4) oraz posiada trzy magazyny (W1 – W3) przechowujące niezbędny sprzęt i materiały do naprawy każdego typu awarii. W mieście zlokalizowano 4 budynki zaliczane do infrastruktury krytycznej (IK1 – IK4). Na podstawie informacji o rodzaju uszkodzenia sieci wodociągowej *Algorytm 4* dokonał następującej klasyfikacji:

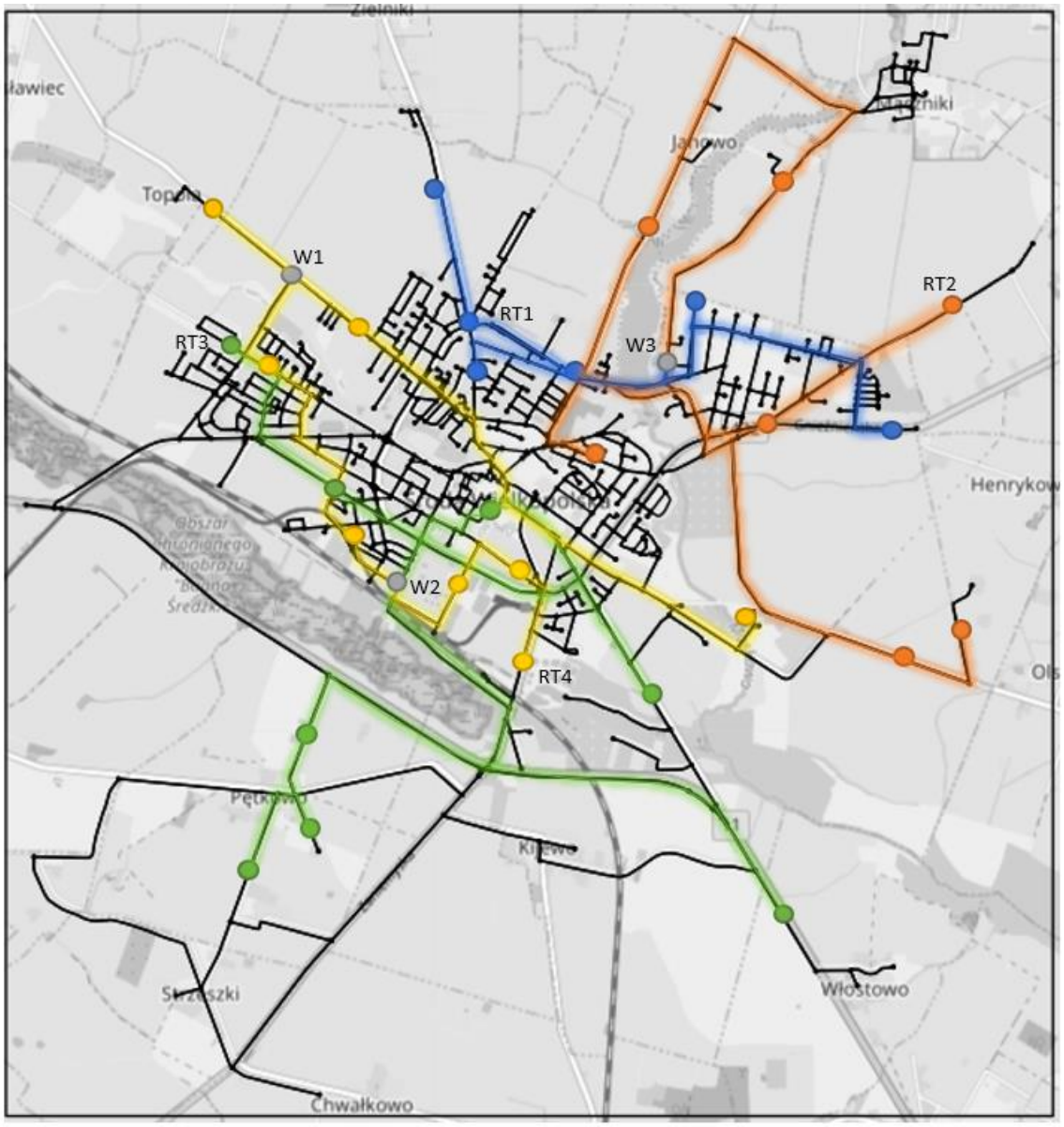
- **kategoria C1**: F4, F7, F17, F20, F22;
- **kategoria C2**: F1, F6, F12, F13, F15, F16, F18, F26, F27, F28;
- **kategoria C3**: F2, F3, F5, F8, F9, F10, F11, F14, F19, F21, F23, F24, F25.

Na rys. 8.12 przedstawiono miejsce występowania awarii, lokalizację magazynów, budynków infrastruktury krytycznej oraz ekip naprawczych.



Rys. 8.12 Scenariusz testowy Środa Wlkp. – Scenario_1 [opracowanie własne]

Algorytm priorytetyzacji wskazał awarię o identyfikatorze *F18* jako awarię priorytetową, ponieważ uniemożliwia transport wody do budynku infrastruktury krytycznej *IK4* oraz awarię *F17*, ponieważ blokuje transport wody do budynku o identyfikatorze *IK3*. *Algorytm 6* zaproponował natomiast agregację następujących awarii: *F19* z *F20*, *F26* z *F28* oraz *F23* z *F27* w celu zaoszczędzenia sumarycznej liczby zasuw do zamknięcia. Na rys. 8.13 przedstawiono rezultat algorytmu szeregowania zadań z uwzględnieniem agregacji. Kolorem niebieskim oznaczono trasę ekipy RT1, pomarańczowym RT2, zielonym RT3 natomiast żółtym RT4.



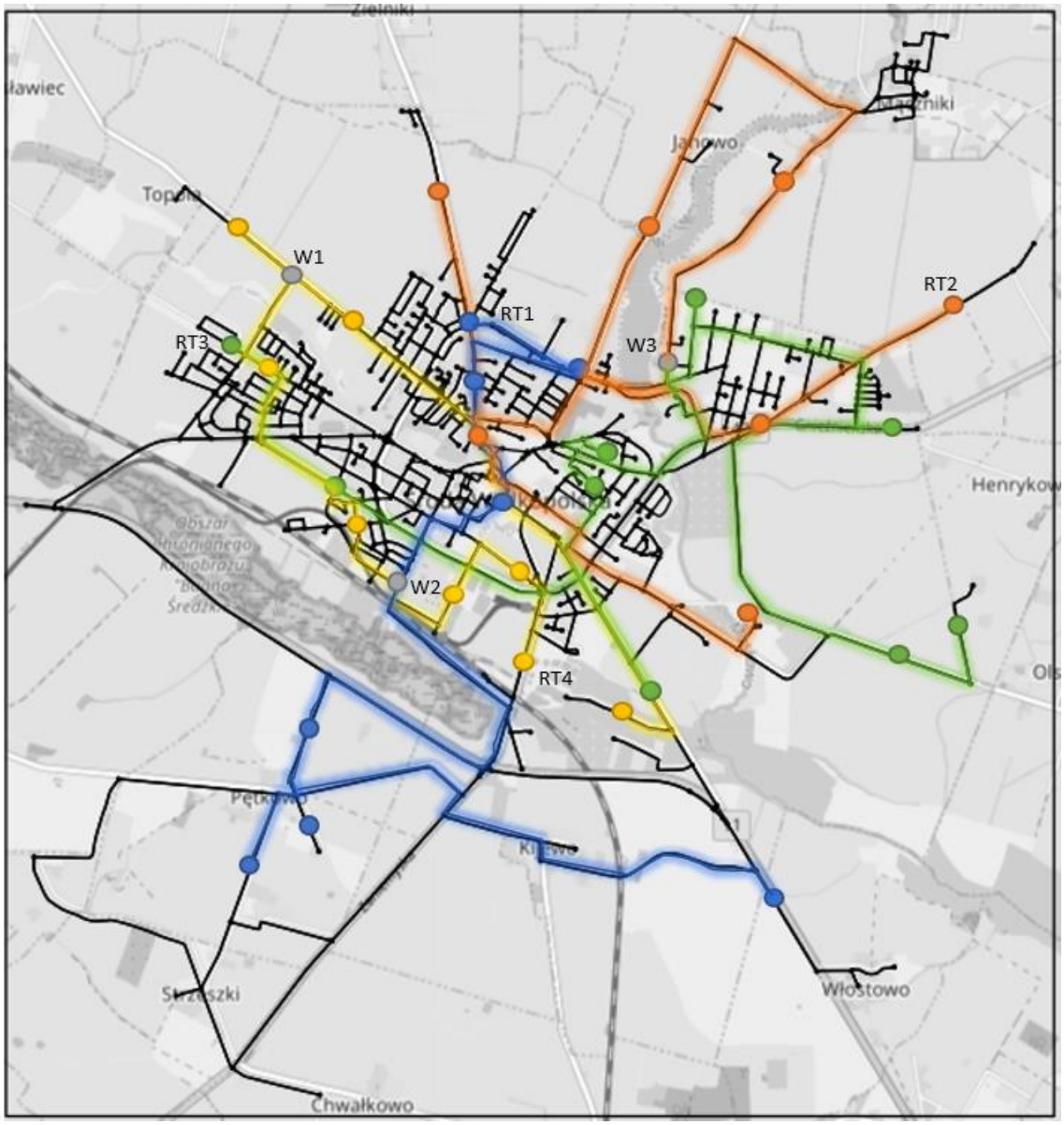
Rys. 8.13 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_1 (z agregacją) [opracowanie własne]

Całkowity prawdopodobny czas naprawy wszystkich awarii (nie wliczając dojazdu) wyniósłby około 18 dni 3 godzin i 38 minut. W przypadku *Algorytmu 7* wyznaczającego trasę poszczególnym ekipom naprawczym (z uwzględnieniem agregacji) otrzymano następującą propozycję:

- **RT1**: F22, F26_F28, F9, W3, F21, F5;
- **RT2**: F17, F4, F6, W3, F23_F27, W3, F11, F10;

- **RT3**: F7, F19_F20, F12, W2, F3, F14, F2, F8;
- **RT4**: F18, F13, W2, F15, F1, W1, F16, F25, W1, F24.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 4 dni, 16 godzin i 9 minut, RT2 – 4 dni, 15 godzin i 3 minuty, RT3 – 4 dni 16 godzin i 8 minut oraz RT4 – 4 dni i 19 godzin. Na rys. 8.14 przedstawiono rezultat algorytmu szeregowania zadań bez uwzględniania agregacji.



Rys. 8.14 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_1 (bez agregacji) [opracowanie własne]

Na rys. 8.14 kolorem niebieskim oznaczono trasę ekipy RT1, pomarańczowym RT2, zielonym RT3 natomiast żółtym RT4. Wersja algorytmu wyznaczania tras nieuwzględniającego agregacji zaproponowała następujące rozdysponowanie zleceń naprawczych.

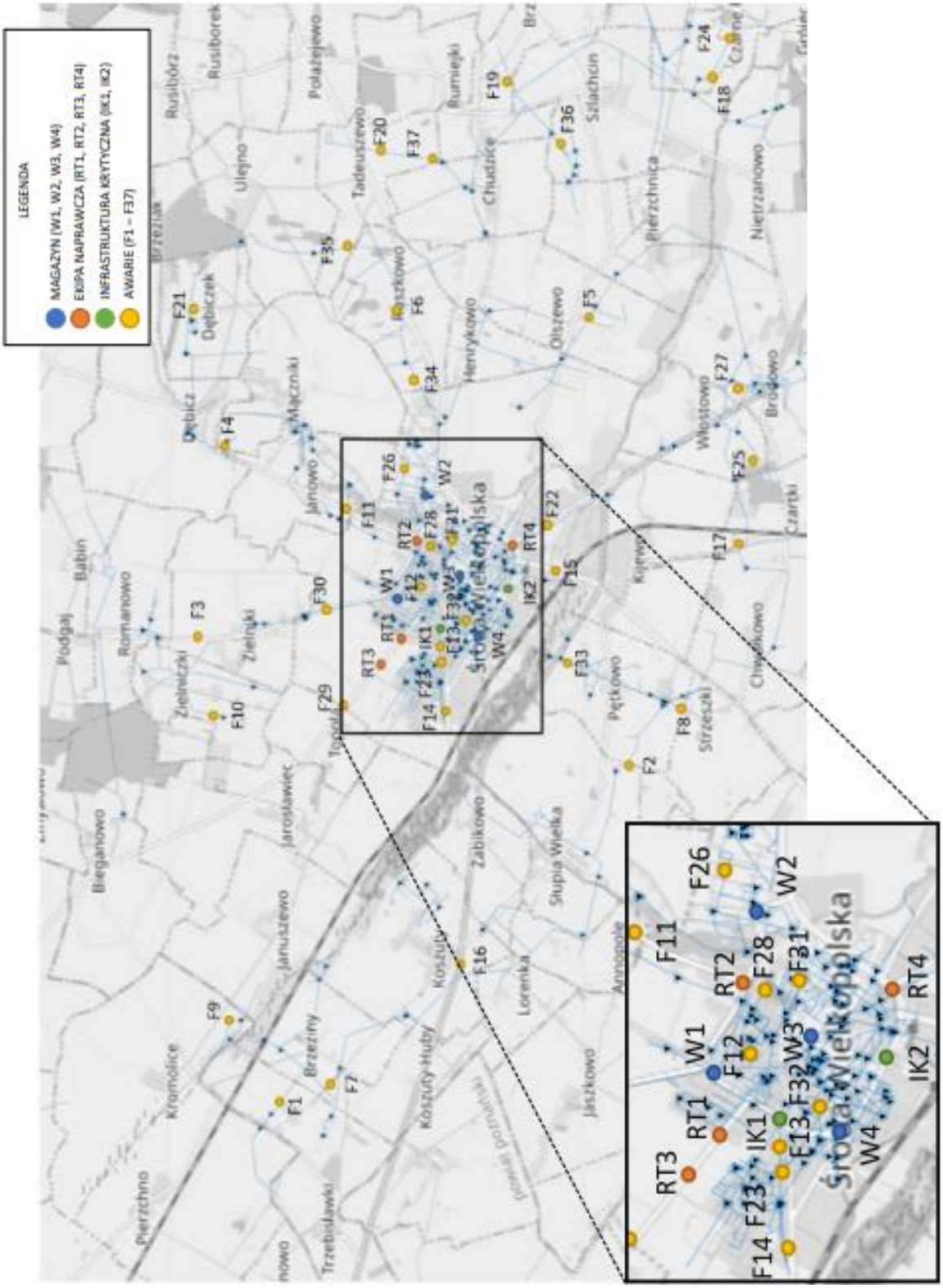
- **RT1**: F22, F26, F12, W2, F3, F14, F2, F8;
- **RT2**: F17, F4, F6, W3, F28, F9, F24;
- **RT3**: F7, F20, F27, F23, W3, F21, F5, F11, F10;
- **RT4**: F18, F13, W2, F15, F1, F16, F25, W1, F19.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 4 dni, 8 godzin i 33 minuty, RT2 – 4 dni, 23 godziny i 26 minut, RT3 – 5 dni, 9 godzin i 5 minut oraz RT4 – 4 dni 15 godzin i 16 minut.

8.4.2.2 Scenariusz drugi (Scenario_2)

Drugim analizowanym scenariuszem testowym (Scenario_2) jest przypadek wystąpienia wielu awarii, w którym sumaryczna liczba awarii wyniosła 37 (10 awarii typu F1, 15 awarii typu F2 oraz 12 awarii typu F3). Analogicznie do przykładu pierwszego liczba dostępnych ekip remontowych wynosi cztery, a liczba budynków infrastruktury krytycznej dwa. W tym przypadku testowym dostępne są cztery magazyny, których lokalizacja została wylosowana. W analizowanym przypadku poza miastem Środa Wielkopolska założono możliwość wystąpienia awarii w okolicach miasta (do których wodę dystrybuuje to samo przedsiębiorstwo). Na rys. 8.15 przedstawiono miejsce występowania awarii, lokalizację magazynów, budynków infrastruktury krytycznej oraz ekip naprawczych. Na podstawie informacji o rodzaju uszkodzenia sieci wodociągowej *Algorytm 4* dokonał następującej klasyfikacji na kategorie:

- **C1**: F13, F14, F23, F32;
- **C2**: F8, F12, F15, F16, F17, F18, F22, F26, F27, F28, F30, F31, F33;
- **C3**: F1, F2, F3, F4, F4, F6, F7, F9, F10, F11, F19, F20, F21, F24, F25, F29, F34, F35, F36, F37.



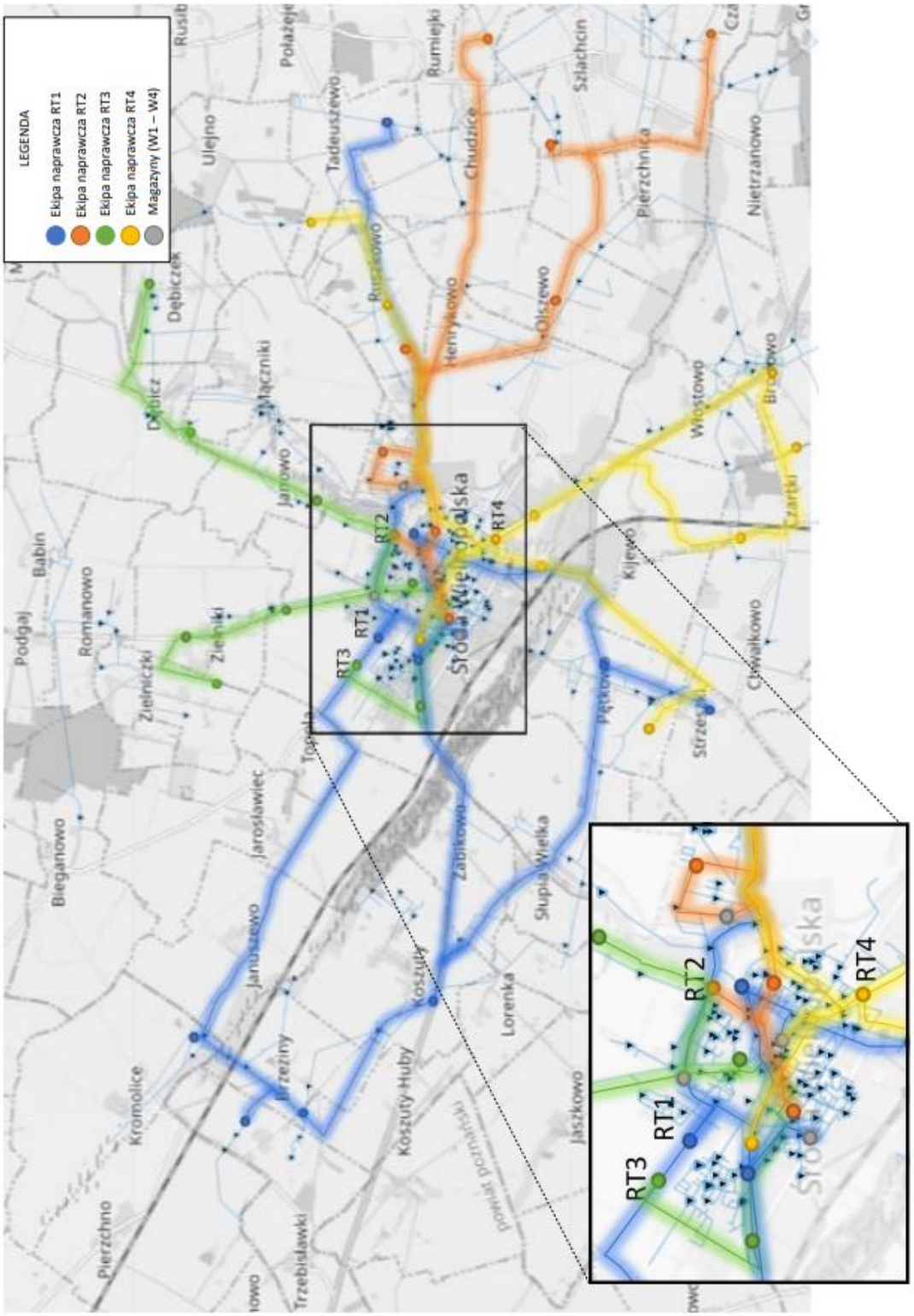
Rys. 8.15 Scenariusz testowy Środa Wlkp. – Scenario_2 [opracowanie własne]

Algorytm priorytetyzacji wskazał awarie o identyfikatorach *F13, F14* oraz *F23*. Stanowią one poważne zagrożenie dla funkcjonowania sieci oraz uniemożliwiają transport wody do budynku infrastruktury krytycznej *IK1*. *Algorytm 6* zaproponował natomiast agregację następujących awarii: *F18* z *F24, F20* z F37 oraz *F29* z *F30*. Na rys. 8.16 przedstawiono rezultat algorytmu szeregowania zadań z agregacji.

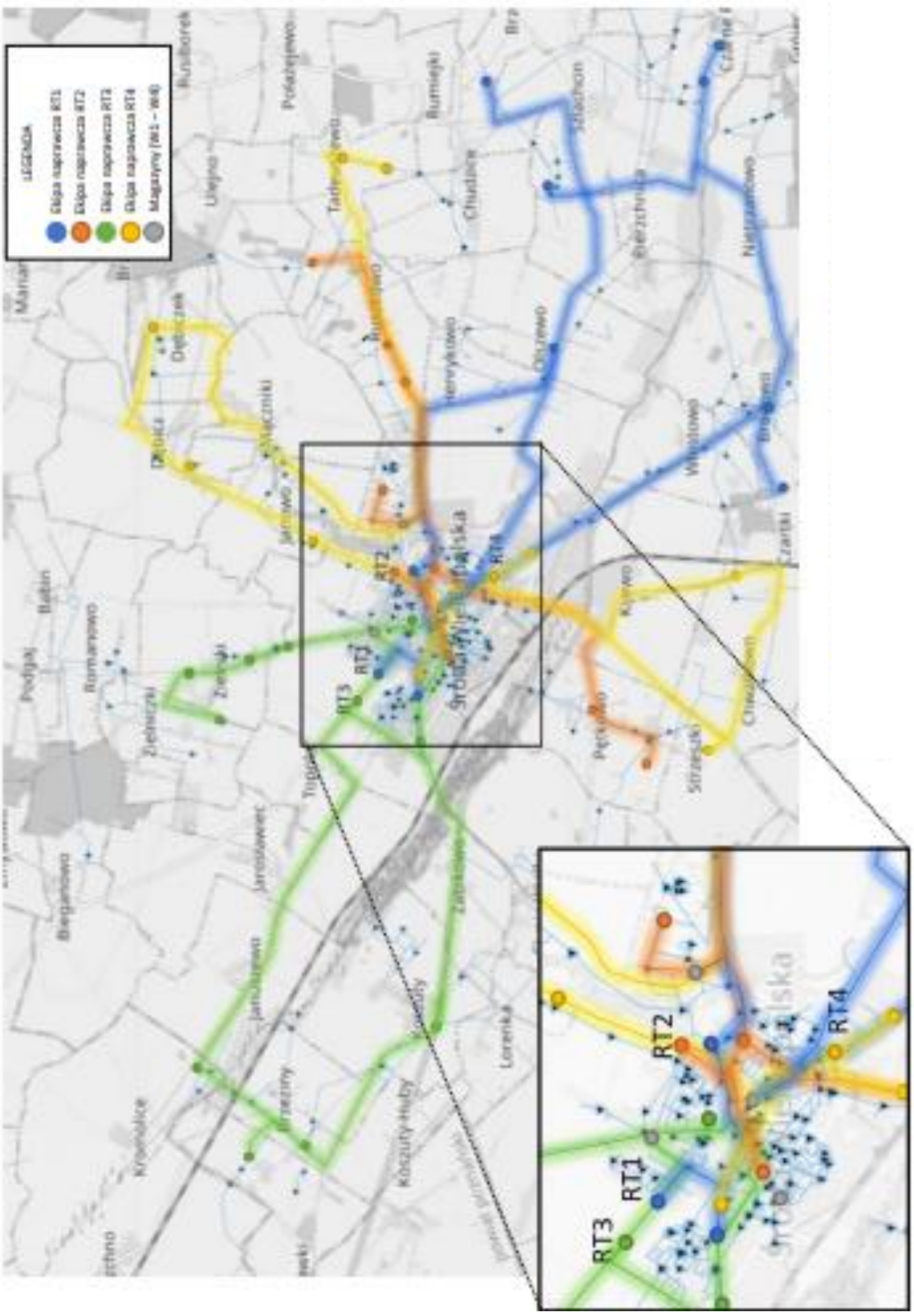
Całkowity prawdopodobny czas naprawy wszystkich awarii (nie wliczając dojazdu) wyniósłby około 20 dni 17 godzin i 40 minut. W przypadku *Algorytmu 7* wyznaczającego trasę poszczególnym ekipom naprawczym (z uwzględnieniem agregacji) otrzymano następującą propozycję:

- **RT1**: F23, F28, W3, F33, F8, F16, W4, F7, F1, F9, W1, F20_F37;
- **RT2**: F32, F31, W2, F26, F18_F24, F36, W2, F5, F34, F19;
- **RT3**: F14, F12, W1, F29_F30, F3, F10, W1, F11, F4, F21;
- **RT4**: F13, F22, F15, W3, F27, F17, F25, W3, F2, F6, F35.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 5 dni, 6 godzin i 22 minuty, RT2 – 5 dni, 7 godzin i 34 minuty, RT3 – 5 dni 6 godzin i 55 minut oraz RT4 – 4 dni, 20 godzin i 48 minut.



Rys. 8.16 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_2 (z agregacją) [opracowanie własne]



Rys. 8.17 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_2 (bez agregacji) [opracowanie własne]

Na rys. 8.17 przedstawiono rezultat algorytmu szeregowania zadań bez uwzględniania agregacji. Kolorem niebieskim oznaczono trasę ekipy RT1, pomarańczowym RT2, zielonym RT3 natomiast żółtym RT4. Wersja algorytmu wyznaczania tras nieuwzględniającego agregacji zaproponowała następujące rozdysponowanie zleceń naprawczych:

- **RT1**: F23, F28, W2, F18, F27, F25, W3, F5, F24, F36, F19;
- **RT2**: F32, F31, W2, F26, F33, F2, W3, F34, F6, F35;
- **RT3**: F14, F12, W1, F30, W1, F16, F7, F1, F9, W1, F29, F3, F10;
- **RT4**: F13, F22, F15, W3, F17, F8, W3, F11, F4, F21, F20, F37.

Prawdopodobne czasy naprawy poszczególnych ekip prezentują się następująco: RT1 – 4 dni, 23 godziny i 22 minuty, RT2 – 5 dni, 5 godzin i 22 minuty, RT3 – 5 dni, 8 godzin i 29 minut oraz RT4 – 5 dni 6 godzin i 24 minuty.

8.5 Poprawność działania systemu i analiza wyników

W zależności od rozpatrywanego algorytmu sposób weryfikacji poprawności wyników różnił się. W przypadku algorytmów klasyfikujących elementy wchodzące w skład sieci wodociągowej (*Algorytm 1*) oraz wyznaczania szlaku wody z analizą przepływów (*Algorytm 2*) wykorzystano oprogramowanie EPANET. Dla wybranego testowego modelu hydraulicznego przygotowano zbiór scenariuszy testowych, które zostały uruchomione z wykorzystaniem zaimplementowanych rozwiązań oraz środowiska EPANET. Jednoznaczność wyników we wszystkich przypadkach potwierdziła poprawność implementacji zaproponowanych rozwiązań.

Algorytm wyznaczający sektory sieci wodociągowej oraz określający zbiór minimalnej liczby zasuw do zamknięcia (*Algorytm 3*) został sprawdzony w sposób ręczny. Na podstawie zbioru losowych identyfikatorów przewodów i wiedzy o lokalizacji zasuw autor pracy wyznaczył segmenty i zasuwy do zamknięcia. Wyniki autora porównano z wynikami otrzymanymi przez algorytm. Wyniki były identyczne we wszystkich przypadkach. Aby określić minimalny zbiór zasuw do zamknięcia wykorzystano ponownie oprogramowanie EPANET, w którym to autor pracy

przetestował dla wybranych segmentów wszystkie możliwe kombinacje zamknięć. Analiza wyników potwierdziła poprawność działania zaproponowanego rozwiązania.

Algorytm klasyfikacji awarii (*Algorytm 4*), priorytetyzacji (*Algorytm 5*) oraz agregacji awarii (*Algorytm 6*) zostały przetestowane w sposób empiryczny wraz z ekspertem zajmującym się tematyką poruszaną w niniejszej pracy. Na podstawie wybranych modeli testowych oraz losowo utworzonego zbioru awarii różnego typu algorytmy oraz ekspert mieli za zadanie utworzyć listę, której porządek determinowałaby kolejność ich naprawy. Równocześnie ekspert na podstawie informacji o infrastrukturze krytycznej wyznaczyć miał awarie priorytetowe oraz wskazać te, które można by zagregować. Stopień podobieństwa rozwiązań wyznaczenia kolejności oscylował w granicy 92%, co według autora pracy można uznać za poprawne.

W zależności od stopnia złożoności analizowanego modelu hydraulicznego (liczba węzłów, przewodów i innych elementów infrastruktury) czas trwania obliczeń różnił się. Najwięcej czasu potrzebowały algorytm klasyfikujący elementy sieci oraz algorytm wyznaczający segmenty. Dla analizowanych przykładów czas wykonywania algorytmów był następujący:

- *Algorytm 1:*
    - **Net3:** 3 minuty 24 sekundy;
    - **Środa Wielkopolska:** 56 minut i 41 sekund.
- *Algorytm 3*:
    - **Net3:** 6 minut 29 sekund;
    - **Środa Wielkopolska:** 2 godziny, 42 minut i 16 sekund.

Należy nadmienić, że powyższe algorytmy nie są wykonywane w trakcie sytuacji katastroficznej. Powyższe algorytmy należy uruchamiać w sytuacji modernizacji sieci w celu uaktualnienia danych o elementach krytycznych i segmentach. Ważna jest również informacja o stanie zasuw. Często zdarza się tak, że po dotarciu ekipy na miejsce występowania zasuwy okazuje się, że jest uszkodzona co uniemożliwia jej otwarcie lub zamknięcie. Czas trwania algorytmu klasyfikacji jest uzależniony od liczby

analizowanych przypadków. W Tabeli 8.2 przedstawiono wyniki działania zbioru zaproponowanych algorytmów w analizowanych przypadkach testowych.

Tab. 8.2 *Informacje o scenariuszach testowych – informacje podstawowe [opracowanie własne]*

Wyniki zbiorcze scenariuszy katastroficznych – informacje podstawowe									
ID SIECI		**Net3**				**Środa WLKP**			
		Scenario_1		Scenario_2		Scenario_1		Scenario_2	
Liczba awarii typu F1		5		3		8		10	
Liczba awarii typu F2		5		7		10		15	
Liczba awarii typu F3		5		8		10		12	
Liczba ekip remontowych		3		3		4		4	
Liczba magazynów		2		3		3		4	
Liczba budynków krytycznych		2		2		4		2	
Proces agregacji		TAK	NIE	TAK	NIE	TAK	NIE	TAK	NIE
Czas trwania symulacji *Algorytmu 7*		00:01:49	00:02:04	00:02:06	00:02:11	00:21:08	00:23:49	00:38:44	00:42:15
Sumaryczny czas przejazdu [HH:mm:ss]	**RT1**	00:37:23	00:32:48	00:20:54	00:25:32	00:09:15	00:12:36	00:41:54	00:39:54
	RT2	00:12:11	00:21:59	00:27:35	00:19:38	00:18:30	00:16:55	00:46:12	00:25:31
	RT3	00:28:41	00:20:32	00:10:47	00:11:25	00:19:01	00:14:54	00:21:12	00:29:27
	RT4	-	-	-	-	00:13:25	00:15:32	00:39:55	00:30:47
Przebyty dystans [km]	**RT1**	24.3	21.9	13.8	17.1	8.9	13.85	48.74	47.05
	RT2	7.7	14.5	18.5	13.1	17.67	16:25	40.25	29.45
	RT3	18.4	13.8	6.9	7.1	20.54	17.14	20.17	34.49
	RT4	-	-	-	-	12.31	15.11	46.61	34.98
Całkowity czas naprawy awarii		9 dni, 11 godz. 3 min.		11 dni, 3 godz. 5 min.		18 dni, 3 godz. 38 min.		20 dni, 17 godz. 40 min,	
Liczba zleconych awarii	**RT1**	F1: **2** F2: **1** F3: **3**	F1: **2** F2: **2** F3: **1**	F1: **0** F2: **2** F3: **3**	F1: **1** F2: **2** F3: **3**	F1: **1** F2: **1** F3: **4**	F1: **3** F2: **2** F3: **2**	F1: **4** F2: **3** F3: **3**	F1: **1** F2: **5** F3: **3**
	RT2	F1: **1** F2: **2** F3: **1**	F1: **2** F2: **1** F3: **3**	F1: **3** F2: **2** F3: **2**	F1: **2** F2: **2** F3: **2**	F1: **2** F2: **3** F3: **2**	F1: **2** F2: **2** F3: **2**	F1: **1** F2: **3** F3: **5**	F1: **2** F2: **0** F3: **6**
	RT3	F1: **2** F2: **2** F3: **1**	F1: **1** F2: **2** F3: **1**	F1: **0** F2: **3** F3: **3**	F1: **0** F2: **3** F3: **3**	F1: **4** F2: **2** F3: **2**	F1: **2** F2: **2** F3: **4**	F1: **3** F2: **4** F3: **2**	F1: **5** F2: **3** F3: **2**
	RT4	-	-	-	-	F1: **1** F2: **4** F3: **2**	F1: **1** F2: **4** F3: **2**	F1: **2** F2: **5** F3: **2**	F1: **2** F2: **7** F3: **1**
Sumaryczny czas naprawy zleconych awarii [HH/mm]	**RT1**	70/22	74/28	91/26	95/17	110/09	104/33	126/22	119/27
	RT2	71/44	74/09	95/32	92/57	109/03	115/26	127/34	125/22
	RT3	75/20	71/24	102/2	104/16	110/58	122/05	126/55	128/29
	RT4	-	-	-	-	113/00	111/16	116/48	126/24
Liczba odwiedzonych magazynu		5	4	5	4	7	5	9	9
Liczba zamkniętych zasuw		31	37	40	54	72	83	106	117

Na podstawie otrzymanych wyników (Tabela 8.2) zauważyć można, że zaproponowane rozwiązanie rozdysponowuje zlecenia w taki sposób, aby sumaryczne czasy napraw poszczególnych ekip były jak najbardziej zbliżone. Proces klasyfikacji i priorytetyzacji działa poprawnie wskazując ekipom identyfikatory tych awarii, które

należy usunąć w pierwszej kolejności. W przypadku procesu agregacji można zauważyć zmniejszenie sumarycznej liczby zasuw do zamknięcia średnio o: 17% (Net3 – Scenario_1), 26% (Net3 – Scenario_2), 14% (Środa Wlkp. – Scenario_1) oraz 10% (Środa Wlkp. – Scenario_2). W Tabeli 8.3 przedstawiono zestawienie zużytych materiałów naprawczych w analizowanych scenariuszach.

Tab. 8.3 *Informacje o scenariuszach testowych – zużycie materiałów [opracowanie własne]*

Wyniki zbiorcze scenariuszy katastroficznych – zużycie materiałów										
ID sieci		**Agregacja**	**Identyfikator ekipy**	**Zaciski**			**Przewody**			**Inne materiały**
				D1	**D2**	**D3**	**D1**	**D2**	**D3**	
Net3	Scenario_1	TAK	RT1	-	-	4	2	-	4	95
			RT2	-	-	10	-	2	-	60
			RT3	6	-	6	-	2	-	70
		NIE	RT1	-	-	12	2	-	-	70
			RT2	-	-	8	-	2	4	95
			RT3	6	-	4	-	2	-	60
	Scenario_2	TAK	RT1	4	-	4	-	-	6	90
			RT2	2	-	12	-	-	4	100
			RT3	-	8	4	2	-	4	105
		NIE	RT1	4	-	6	-	-	6	100
			RT2	2	-	10	-	-	4	90
			RT3	-	8	4	2	-	4	105
Środa WLKP	Scenario_1	TAK	RT1	-	4	-	4	4	-	95
			RT2	2	4	4	2	2	-	100
			RT3	2	8	8	2	2	-	135
			RT4	-	12	2	-	2	2	110
		NIE	RT1	-	6	4	2	2	-	105
			RT2	-	8	-	2	2	2	105
			RT3	2	6	4	2	4	-	120
			RT4	2	8	6	2	2	-	110
	Scenario_2	TAK	RT1	4	6	10	-	4	2	145
			RT2	-	6	8	2	6	2	155
			RT3	4	14	4	-	4	-	130
			RT4	4	12	8	-	2	2	135
		NIE	RT1	4	6	12	-	4	2	145
			RT2	-	4	-	2	8	2	140
			RT3	4	12	8	-	4	-	135
			RT4	4	16	10	-	-	2	145

Weryfikacja stanu i zużycia materiałów pozwoliła określić poprawność implementacji metod odpowiedzialnych za analizę stanów magazynowych i pojazdów dostępnych ekip. Jak można zauważyć w Tabeli 8.3 sumaryczna liczba zużytych materiałów w poszczególnych scenariuszach jest identyczna (z i bez agregacji). Pozwala to na stwierdzenie, że sposób poboru materiałów oraz ich doboru z magazynów jest poprawny oraz warunki określające możliwość naprawy określonych awarii również zostały należycie zaimplementowane.

9. Wnioski

9.1 Podsumowanie

Niniejsza rozprawa miała na celu opracowanie metodyki określania kolejności zadań naprawczych dla ekip remontowych w sytuacji wystąpienia wielu awarii w złożonym sieciowym systemie dystrybucji. Zdefiniowano zadanie szeregowania, przy założeniu z góry określonej liczby i lokalizacji awarii oraz dostępnych informacji o topologii sieci oraz położenia magazynów i dostępności ekip naprawczych. Metodę rozwiązania problemu zaprezentowano w odniesieniu do systemu zaopatrzenia w wodę. Zaproponowanym rozwiązaniem jest grupa algorytmów pozwalających na uzyskanie najkrótszego czasu niezbędnego do przywrócenia ciągłości dystrybucji wody przy uwzględnieniu zadanych kryteriów oraz ograniczeń.

Dokonano przeglądu literatury w zakresie zarządzania procesami dystrybucji wody, modelowania pracy systemów zbiorowego zaopatrzenia w wodę, rodzajów awarii, sposobów napraw awarii, szeroko rozumianej analizy krytyczności, algorytmów szeregowania zadań oraz dostępnych narzędzi informatycznych. Przeprowadzona analiza pozwoliła na określenie kroków koniecznych do realizacji systemu wsparcia operatora rozdzielającego zlecenia naprawcze. Zgodnie z wprowadzoną metodologią, niezbędnym elementem w procesie szeregowania zadań jest identyfikacja poszczególnych elementów infrastruktury sieciowej oraz jej wpływu na przebieg procesu. Odwzorowanie powyższych informacji w bazie wiedzy pozwoliło na lepszą klasyfikację zgłoszonych awarii oraz określenie identyfikatorów elementów (zasuw) niezbędnych w procesie przywracania lub odseparowania uszkodzonego elementu ze struktury sieci.

Analiza doświadczeń eksploratora sieci wodociągowej i eksperta w zakresie poruszanej tematyki umożliwiła zidentyfikowanie ograniczeń i kryteriów, które należy uwzględnić w procesie decyzyjnym. Na ich podstawie zaproponowano oraz zaimplementowano algorytmy analizujące i określające rodzaje awarii, dostępność zasobów naprawczych, możliwość agregacji czy priorytetyzację.

Podsumowując, zaproponowane rozwiązanie składa się z 7 algorytmów:

- Algorytm 1: odpowiada za klasyfikację elementów wchodzących w skład sieci wodociągowej;
- Algorytm 2: wyznacza trasę przepływu wody od źródła do celu analizując dodatkowo przepływy na podstawie symulacji hydraulicznych;
- Algorytm 3: odpowiada za sektoryzację sieci, wyznaczanie zasuw niezbędnych do odizolowania danego sektora;
- Algorytm 4: odpowiada za klasyfikację zbiorów dostępnych awarii;
- Algorytm 5: w oparciu o dane lokalizacyjne budynków infrastruktury krytycznej, odpowiada za przydzielanie priorytetów poszczególnym awariom;
- Algorytm 6: na podstawie informacji o przynależności awarii do poszczególnych segmentów proponuje ich agregację w celu minimalizacji sumarycznej liczby zasuw do zamknięcia. Algorytm ten proponuje dwa tryby agregacji: pierwszy w sytuacji, gdy awarię znajdują się w tym samym segmencie oraz drugi w sytuacji gdy awarie znajdują się w segmentach posiadających wspólny element (zasuwę);
- Algorytm 7: odpowiada za analizę danych otrzymanych w wyniku pracy wcześniej wspomnianych algorytmów. Na ich podstawie oraz lokalizacji magazynów i dostępnych ekip wyznacza zlecenia naprawcze.

W pracy przedstawiono wyniki badań, w których zaproponowaną metodykę zastosowano dla przykładowych sytuacji decyzyjnych. Wymagało to utworzenia środowiska symulacyjnego, w którym algorytmy zostały zaimplementowane w postaci skryptu komputerowego. W tym celu wykorzystano środowisko PyCharm i język programowania Python, za pośrednictwem którego utworzono bazę danych w plikach .JSON przechowującą informację o analizowanych sieciach wodociągowych. W celu integracji modelu hydraulicznego ze zrealizowanymi skryptami wykorzystano ogólnodostępną bibliotekę WNTR wykorzystującą silnik obliczeń EPANET.

9.2 Wnioski końcowe

Poprawa efektywności procesu przywracania ciągłości dystrybucji wody w sytuacji pokatastroficznej poprzez szeregowanie zadań dostępnych ekip naprawczych nie należy do zadań prostych. Złożoność topologii sieci wodociągowej, zmienność warunków pracy oraz mnogość informacji o stanie obiektu w danej chwili stanowią trudność realizacji zadania szeregowania wyłącznie w oparciu o wiedzę osoby odpowiedzialnej za przekazywanie zleceń naprawczych. W sytuacjach katastroficznych niezbędne jest wykorzystanie dostępnych narzędzi informatycznych wspomagających pracę decydentów. Niniejsza praca wskazuje, że wykorzystanie narzędzi informatycznych w połączeniu z wiedzą ekspercką może stanowić skuteczną strategię minimalizacji czasu przywracania pełnej zdolności sieci wodociągowej w sytuacji pokatastroficznej. W oparciu o przeprowadzone analizy i uzyskane wyniki można sformułować następujące wnioski:

(1) Uzyskane wyniki uzasadniają postawioną tezę, iż ***dla zdefiniowanego problemu decyzyjnego możliwe jest określenie kryteriów wyboru, ograniczeń, które pozwolą zachować największą sprawność systemu wodociągowego w sytuacji wystąpienia wielu awarii.***

(2) Zaproponowana grupa algorytmów pozwala na automatyczne budowanie scenariuszy testowych oraz analizę działania sieci w sytuacjach wystąpienia wielu awarii, co pozwoli na przeciwdziałanie i niwelowanie punktów krytycznych występujących w analizowanym przypadku.

(3) Wyniki uzyskane w przeprowadzonych scenariuszach testowych potwierdziły poprawność zaimplementowanych rozwiązań.

(4) Integracja rozwiązania z istniejącymi systemami typu SCADA umożliwia aktualizację bazy wiedzy dotyczącej topologii sieci jak i stanu urządzeń wykonawczych (np. zasuw).

(5) Opracowana metodyka pozwala skrócić wymagany czas na poznanie możliwych rozwiązań taktycznych dotyczących dysponowania zleceń naprawczych.

(6) Przedstawione rozwiązania mogą zostać wykorzystane do analizy innych złożonych systemów takich jak system dystrybucji gazu, ropy czy systemu elektroenergetycznego.

W analizie pracy konieczne stało się również uwydatnienie charakteru praktycznego (na przykładzie systemów zbiorowego zaopatrzenia w wodę), ponieważ ogólnikowe rozważania uniemożliwiłyby potwierdzenie przedstawionej tezy oraz wykazania szczegółów opracowanej metody. Przeprowadzona w pracy analiza i uzyskane wyniki pozwalają na określenie ogólnego schematu (etapów) postępowania w odniesieniu do innych złożonych systemów dystrybucyjnych:

1. **Etap 1** – w którym na podstawie modelu matematycznego obiektu oraz wyników symulacji jego pracy następuje dynamiczna identyfikacja elementów wchodzących w jego skład.
2. **Etap 2** – odpowiedzialny jest za analizę danych oraz właściwą klasyfikację awarii (z uwzględnieniem liczby uszkodzonych elementów oraz infrastruktury krytycznej).
3. **Etap 3** – w którym na podstawie dostarczonych danych zostaje wyznaczona propozycja rozwiązania problemu szeregowania zadań (marszruta dla poszczególnych ekip naprawczych).

Zaproponowane rozwiązanie wymaga aktualnej wiedzy o analizowanym obiekcie. Dostarczony model hydrauliczny reprezentujący wskazaną sieć powinien być skalibrowany i poprawnie zaprojektowany pod względem wymagań topologicznych niezbędnych do uruchomienia symulacji. Poprawność działania systemu uzależniona jest bowiem od wiarygodności posiadanych informacji o obiekcie. Na podstawie informacji otrzymanych od ekspertów dziedzinowych zajmujących się tematyką poruszaną w niniejszej pracy (w tym służb eksploatacyjnych *Miejskiego Przedsiębiorstwa Energetyki Cieplnej Wodociągów i Kanalizacji w Środzie Wielkopolskiej*) sformułowano najważniejsze elementy niezbędne w procesie decyzyjnym dotyczącym typowania awarii i rozdysponowywania zleceń naprawczych. Zdobyte informacje pozwoliły na opracowanie algorytmów, które mogą pełnić rolę systemu eksperckiego lub wspomagania decyzji.

Warto nadmienić, że opracowana metoda może być rozwijana. Istnieje wiele możliwości rozwoju zaproponowanych algorytmów począwszy od optymalizacji i refaktoryzacji kodu, skutkiem którego byłaby szybsza informacja zwrotna. Ponadto warto rozpatrzeć każdy algorytm z osobna i zaproponować bardziej złożone odwzorowanie rzeczywistości np. poprzez dodanie większej liczby materiałów i narzędzi naprawczych.

Zaproponowane rozwiązanie zakłada z góry znaną i określoną liczbę awarii co w rzeczywistości nie zawsze ma miejsce. Warto zastanowić się nad implementacją gotowych rozwiązań, zadaniem których byłoby wyszukiwanie wycieków i zgłaszanie prawdopodobnych miejsc awarii.

Dzięki modułowości rozwiązania istnieje możliwość wykorzystania opracowanej metody przez inne systemy. Przykładowo wykorzystując otwarte środowisko Python oraz bibliotekę PyPSA oraz PandaPipes istnieje możliwość opracowania i implementacji dodatkowych modułów odpowiedzialnych za modelowania systemów energetycznych oraz gazowych. Tego typu moduły wykorzystujące zaprezentowane algorytmy mogłyby w sposób analogiczny do systemu dystrybucji wody zwracać informacje o kolejności napraw awarii w ww. systemach.

Bibliografia

[1] Abdelhafid M., Fourati M., Fourati C., A genetic algorithm-based intelligent solution for water pipeline monitoring system in a transient state, WIley Special issue paper, 2020, DOI: https://doi.org/10.1002/cpe.5959.

[2] Antonowicz A., System sterowania procesami biodegradacji zanieczyszczeń w środowisku wodnym, Praca magisterska, Politechnika Poznańska, Poznań 2016.

[3] Antonowicz A., Systemy Wspomagania Decyzji w gospodarce wodno-ściekowej w: Współczesne aspekty badawcze. Gospodarka-Świat-Człowiek. Aspekty teoretyczno-praktyczne badań naukowych. Część V (red. Joanna Nowakowska-Grunt, Judyta Kabus), Katowice, Wydawnictwo Naukowe Sophia, 2017, s. 11-19.

[4] Antonowicz A., Bałut A., Urbaniak A., Zakrzewski P., Algorithm for Early Warning System for Contamination in Water Network, 20th International Carpathian Control Conference (ICCC 2019), Kraków-Wieliczka, Poland 2019, s. 1-7.

[5] Antonowicz A., Inda H., Wykorzystanie systemów wbudowanych do sterowania i wizualizacji procesu uzdatniania wody, Praca inżynierska, Państwowa Wyższa Szkoła Zawodowa im. Hipolita Cegielskiego w Gnieźnie, Gniezno 2015.

[6] Antonowicz A., Nowak M., Urbaniak A., Task Scheduling Algorithm for Renovation Teams of Water Distribution Systems, Proceedings of the 2020 21st International Carpathian Control Conference (ICCC): Virtual Conference, Košice, Slovak Republic 2020, 2. 1-6.

[7] Aquanet S.A., Standardy materiałowe obiektów i urządzeń wodociągowych stosowanych na sieciach wodociągowych w obszarze działania, Aquanet S.A., Poznań 2013.

[8] Argent R. M., An overview of model integration for enviromenta application – componetns, frameworks and semantics, Enviromental Modelling & Software, Tom 19, nr 3, 2004, s. 219 -234.

[9] Armando Di Nardo A., Di Natale M., Gisonni C.; Iervolino M., A genetic algorithm for demand pattern and leakage estimation in a water distribution network, Journal of Water Supply: Research and Technology-Aqua, 2015, https://doi.org/10.2166/aqua.2014.004.

[10] Bajer J., Przebinda A., Czynniki wpływające na czas usuwania awarii przewodów wodociągowych i ich uzbrojenia, Gaz Woda i Technika Sanitarna, 11, 2005 s. 20-22.

[11] Bałut A., Brodziak R., Bylka J., Zakrzewski P., Ranking Approach to Scheduling Repairs of a Water Distribution System for the Post-Disaster Response and Restoration Service, Water 2019, s. 1591, DOI: https://doi.org/10.3390/w11081591.

[12] Bąk T. (red.), Współczesne problemy zarządzania nr 10 – Bezpieczeństwo publiczne, EXDRUK, Jarosław 2017, s. 7-8.

[13] Brdyś M., Cimiński A., Drewa M., Kurek W., Sterowanie predykcyjne hydrauliką i jakością wody w SW, Wodociągi - Kanalizacja, nr 11, 2008, s. 38–41.

[14] Brdyś M., Ewald G., Sieć SSWN w oczyszczalni ścieków, Wodociągi i kanalizacja, nr 2, 2008, s. 21-23.

[15] Brodziak R., Synteza scenariuszy eksploatacji i sterowania procesem ujmowania wody metodą sztucznej infiltracji, Rozprawa doktorska, Wydział Budownictwa i Inżynierii Środowiska Politechniki Poznańskiej, Poznań 2017.

[16] Candelieri A., Ponti A., Giordani I., Archetti F., Lost in Optimization of Water Distribution System: Better Call Bayes, preprints.org, 2022 (dostęp: 11.10.2021 g. 12:35).

[17] Cattafi, M., Gavanelli, M., A CLP(FD) program for the optimal placement of valves in a water distribution network, 2011, źródło: http://www.ing.unife.it/docenti/MarcoGavanelli/.

[18] Christodoulou S., Aslani P., Deligianni A., Integrated GIS-Based management of Water Distribution Networks, Joint International Conference on Computing and Decision Making in Civil and Building Engineering, June 14-16, Montréal, Canada, 2006, s. 858-865.

[19] Chrzanowski P., Dudziak S., Kaczmarek J., Monitorowanie sieci wodociągowej pod kątem bezpieczeństwa wody, Praca inżynierska, Wydział Informatyki Politechniki Poznańskiej, Poznań, 2019.

[20] Chudzicki J., Sosnowski S., Instalacje Wodociągowe, „Seidel-Przywecki" SP. z o.o, Warszawa, 2009.

[21] Computers Application in Hydraulic Engineering, wyd. Eighth edition, Bentley Institute Press, Exton, Pa, 2013.

[22] Cosgrove J. W., Loucks P. D., Water management: Current and future challenges and research directions, Water Resources Research vol. 51, Issue 6, 2014, s. 4823-4839 DOI: https://doi.org/10.1002/2014WR016869.

[23] Creaco, E.,Franchini, M., Alvisi, S., Optimal placement of isolation valvesin water distribution systems based on valve cost and weighted average demand shortfall.Journal of Water Resources Planning and Management 24, 15, 2010, s. 4317–4338.

[24] Dudziak A., Charakterystyki przepływowe zasuw wodociągowych, Rynek Instalacyjny, 2008.

[25] Duzinkiewicz K., Zintegrowane sterowanie systemami wodociągowymi – struktury i algorytmy, cz. 1, Wodociągi i Kanalizacje nr 11, 2005, s. 24-26.

[26] Duzinkiewicz K., Piotrowski R., Krause Ł., Nowoczesne sterowanie w oczyszczalniach ścieków, Wodociągi I Kanalizacje, nr 1, 2005.

[27] De Paola F., Galdiero E., Giugni M., Location and setting of valves in water distribution networks using a harmony search approach, Journal of Water Resources Planning and Management 143.6, 2017.

[28] Findeisen W., Analiza systemowa. Podstawy i metodologia, PWN, Warszawa 1985.

[29] Flasiński M., Wstęp do sztucznej Inteligencji Wyd. PWN Warszawa 2011.

[30] Giustolisi O., Savic D. A., Optimal design of isolation valve system for water distribution networks. In Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008, 2008, s. 277-287.

[31] Giustolisi O., Savic D. A., Identification of segments and optimal isolationvalve system design in water distribution networks. Urban Water Journal 7, 2010, s. 1–15.

[32] Halhal D., Walters G., Savic D., Ouazar D., Scheduling of Water Distribution System Rehabilitation Using Structured Messy Genetic Algorithms, Evolutionary Computation, Volume: 7, Issue: 3, Sept. 1999, DOI: 10.1162/evco.1999.7.3.311.

[33] Heidrich Z., Wodociągi i Kanalizacja. Część 1. Wodociągi, WSIP, Warszawa 1999, s. 7.

[34] Hu C., Ren G., Liu C., Li M., Jie W., A Spark-based genetic algorithm for sensor placement in large scale drinking water distribution systems, Springer Cluster Computing volume 20, 2017, s.1089–1099.

[35] Hwandon J., Strategic valve location in a Water Distribution System, Polytechnic Institute and State University in Civil and Enviromental Engineering, Virginia, 2005.

[36] Iwanejko R., Bajer R., Podstawy teoretyczne metody szacowania ryzyka związanego z czasem usuwania awarii sieci wodociągowej, Technical Transaction Environmental Engineering, Wydawnictwo Politechniki Krakowskiej, Zeszyt 1 2011, s. 75 -83.

[37] Klimek M. Algorytmy konstrukcyjne dla problemu harmonogramowania projektu z ograniczonymi zasobami, Zeszyty Naukowe WWSI, No15, Vol. 10, 2016 s.41 -52.

[38] Klise, K.A., Murray, R., Haxton, T., An overview of the Water Network Tool for Resilience (WNTR), In Proceedings of the 1st International WDSA/CCWI Joint Conference, Kingston, Ontario, Canada, July 23-25, 075, 2018.

[39] Klise, K.A., Bynum, M., Moriarty, D., Murray, R., A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study, Environmental Modelling and Software, 95, 420-431, DOI: 10.1016/j.envsoft.2017.06.022, 2017.

[40] Klise, K.A., Hart, D.B., Moriarty, D., Bynum, M., Murray, R., Burkhardt, J., Haxton, T., Water Network Tool for Resilience (WNTR) User Manual, U.S. Environmental Protection Agency Technical Report, EPA/600/R-17/264, 2017.

[41] Koch R., Noworyta A., Procesy mechaniczne w inżynierii chemicznej, Warszawa: Wydawnictwa Naukowo-Techniczne, 1992. ISBN 83-204-1210-2.

[42] Koradecka D., Bezpieczeństwo pracy i ergonomia, Warszawa 1999.

[43] Kowal A., Świderska M., Oczyszczanie Wody, Wyd. Naukowe PWN, Warszawa 2005.

[44] Kowalik P., Stan aktualny i kierunki rozwoju nauki w zakresie teorii i metod optymalizacji systemów wodociągowych i kanalizacyjnych, Opracowanie niepublikowane, 2003.

[45] Kowalski D., Nowe Metody opisu struktur sieci wodociągowych do rozwiązywania problemów ich projektowania i eksploatacji, Monografie Komitetu Inżynierii Środowiska PAN, vol. 88, Lublin 2011

[46] Kummu M., Guillaume J. H. A., Moel H. i inni, The world's road to water scarcity: shortage and stress in the 20th century and pathways towards sustainability, Veldkamp & P. J. Ward, Scientific Reports volume 6, Article number: 38495, 2016.

[47] Kutyłowska M., Metody regresyjne i klasyfikacyjne w analizie i ocenie poziomu awaryjności przewodów wodociągowych, Oficyna wydawnicza Politechniki Wrocławskiej, Wrocław 2019 (ISSN 2657-5035).

[48] Kwiatkowski J., Optymalizacja pracy sieci wodociągowej, Wodociągi i Kanalizacja nr 8, 2007.

[49] Kwietniewski M., Monitorowanie i Geograficzne Systemy Informacji w procesie współczesnej eksploatacji systemów dystrybucji wody i odprowadzania ścieków, Gaz, Woda i Technika Sanitarna, nr 4, 2005, s. 9-14.

[50] Lambert, A., What do we know about pressure-leakage relationships in distribution systems. Proceedings of the IWA Specialised Conference 'System Approach to Leakage Control and Water Distribution Systems Management', Brno, Czech Republic, 2001, May 16-18, 2001, s. 89-96.

[51] Liu H., et al. Failure impact analysis of isolation valves in a water distribution network, Journal of Water Resources Planning and Management 143.7, 2017, s. 235-242.

[52] Łangowski Rafał, Algorytm alokacji punktów monitorowania jakości w systemach dystrybucji wody pitnej, Rozprawa doktorska, Wydział elektrotechniki i automatyki Politechniki Gdańskiej, Gdańsk 2012.

[53] Ławryńczuk M., Computationally Efficient Model Predictive Control Algorithms, http://www.ia.pw.edu.pl/~maciek/badania7.php (30.11.2021 godzina 15:45).

[54] Mercedes M., Gamboa-Medina L., Ribeiro R., Sampling Design for Leak Detection in Water Distribution Networks Procedia Engineering vol. 186, 2017, s. 460-469 https://doi.org/10.1016/j.proeng.2017.03.255.

[55] Milkiewicz F., Chotkowski W., Duzinkiewicz K., Brdyś M., Struktury sterowania systemami wodociągowymi, III Konf.Nauk.-Techn. „Technologia i automatyzacja systemów wodociągowych i kanalizacyjnych TiASWiK'99",Pol. Gdańska, Stawiska, 23-25 czerwca 1999, s. 311-320.

[56] Mitchell G. V., Applying Integrated Urban Water Management Concepts: A Review of Australian Experience, Environmental Management vol. 37, s. 589–605, 2006.

[57] Mulawka J., Systemy Ekspertowe Wyd. Naukowo-Techniczne, Warszawa 1996.

[58] Nonato M., Alvisi S., Optimal Placement of Valves in a Water Distribution Network with CLP(FD), Theory and Practice of Logic Programming 11(4), 2011, s. 731-747, DOI:10.1017/S1471068411000275.

[59] Nowak M., Rozproszone sterowanie, monitorowanie i wizualizacja w inżynierii i ochronie środowiska, w: Inteligentne systemy w inżynierii i ochronie środowiska, red: Nowak M., Urbaniak A. ,Zakład Poligraficzny Moś-Łuczak, Poznań 2007 (s. 13 – 22).

[60] Nowogoński I., Skrócona instrukcja obsługi EPANET, (źródło internetowe), https://www.iis.uz.zgora.pl/files/EPANET-instr.pdf (02.12.2021 godzina 11:00).

[61] Our common Future – Chapter 2: Towards Sustainable Development, Form A/42/427, Report of the World Commission on Environment and Development, 1987.

[62] Pacholski L., Systemy Ekspertowe i Sztuczna Inteligencja, Wyd. Politechniki Poznańskiej, 2012.

[63] Piechurski, F. Efekty wdrożenia monitoringu w wybranej sieci wodociągowej - studium przypadku, Gaz, Woda i Technika Sanitarna, nr 10, 2017 s. 395 - 398

[64] PN-EN 12201 Systemy przewodów rurowych z tworzyw sztucznych do przesyłania wody i do ciśnieniowego odwadniania i kanalizacji -- Polietylen (PE).

[65] PN-EN 1452-2 „Systemy przewodowe z tworzyw sztucznych – Systemy przewodowe z niezmiękczonego poli(chlorku winylu)(PVC-U) do przesyłania wody”

[66] PN-EN 1452-3 „Systemy przewodów rurowych z tworzyw sztucznych do przesyłania wody oraz do ciśnieniowego odwadniania i kanalizacji układanej pod ziemią i nad ziemią – nieplastyfikowany poli(chlorek winylu) (PVC-U) – Część 3: Kształtki”

[67] PN-EN 1092-2 „Kołnierze i ich połączenia. Kołnierze okrągłe do rur, armatury, łączników i osprzętu z oznaczeniem PN. Kołnierze Żeliwne”

[68] Popiołek K (red.)., Kryzysy, katastrofy, kataklizmy: Zjawiska współczesnej cywilizacji, SPiA, Poznań 2004, s. 26–37.

[69] Rak J., Kwietniewski M., i inni Metody oceny niezawodności i bezpieczeństwa dostawy wody do odbiorców Wyd. Oficyna Rzeszów 2013.

[70] Rak J., Tchórzewska-Cieślak B., Bezpieczeństwo systemów zbiorowego zaopatrzenia w wodę, PAN Instytut Badań Systemowych, Warszawa 2013.

[71] Rak J., Tchórzewska-Cieślak B. Ryzyko w eksploatacji systemów zbiorowego zaopatrywania w wodę, Wyd. 1, Red. Seidel-Przywecki, 2013.

[72] Rauba E., Miłaszewski R., Koszty funkcjonowania systemów zaopatrzenia w wodę, Mat. Konf. Zaopatrzenie w wodę, jakość i ochrona wód TOM 1, Poznań 2012, s.137.

[73] Rojek I., Koncepcja inteligentnego systemu detekcji awarii sieci wodociągowej, Studia i Materiały Polskiego Stowarzyszenia Zarządzania Wiedzą, TOM 37, 2010, s. 264-274.

[74] Rojek I., Studziński J., Algorytmy lokalizacji awarii w sieci wodociągowej przy użyciu sieci neuronowych, Studia i Materiały Polskiego Stowarzyszenia Zarządzania Wiedzą, Tom 8, Bydgoszcz 2011, s. 146- 156.

[75] Rojek I., Studziński J., Sieci neuronowe w lokalizacji awarii w sieci wodociągowej, Studia i Materiały Informatyki Stosowanej, Tom 4, Nr 9, 2012 s. 29-34.

[76] Rojek I. Opracowanie algorytmów lokalizacji awarii sieci wodociągowej w postaci sieci neuronowych. Raport z projektu badawczego Nr 14-0011-10/2010. Warszawa: IBS PAN, 2012 .

[77] Rossman L., Epanet 2 User Manual. National Risk Management Research Laboratory, Cincinnati, 2000.

[78] Rozporządzenie Ministra Zdrowia z dnia 29 marca 2007r. w sprawie jakości wody przeznaczonej do spożycia przez ludzi, źródło: https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20070610417.

[79] Schmidt A., Algorytm szeregowania zadań ekip naprawczych w Inteligentnym Mieście, Praca magisterska, Wydział Informatyki i Telekomunikacji Politechniki Poznańskiej, Politechnika Poznańska, Poznań 2021.

[80] Schiller T., Komputerowe modele sieci wodociągowej, Wodociągi i Kanalizacja, nr 7-8, 2005 s. 26-27.

[81] Sekuła W., Uzdatnianie wody, Wyd. Projprzem-EKO, Bydgoszcz 2000.

[82] Shiddiqi, Ary Mazharuddin, et al. Sensor Placement Strategy to Localize Leaks in Water Distribution Networks with Fluctuating Minimum Night Flow, 13th International Conference on Information & Communication Technology and System (ICTS). IEEE, 2021.

[83] Smutnicki C., Algorytmy szeregowania zadań, Oficyna Wydawnicza Politechniki Wrocławskiej, 2012, Wrocław ISBN: 978-83-7493-713-9 .

[84] Sozański M, Technologia uzdatniania wody – znaczenie, metodologia, perspektywy (w opracowaniu).

[85] Sozański M.M., Sroczan E.M., Urbaniak A., 2001: Application of artificial inteligent methods in computerized system of training wastewater treatment plant operators. Mat. Konf. Nt. :Canadian Society for Civil Engeneering 29th Annual Conference 2001", Victoria, British Columbia, CD-ROM E-21.

[86] Szargut J., Termodynamika. Warszawa: Państwowe Wydawnictwo Naukowe, 1980.

[87] Szpak D., Tchórzewska-Cieślak B., Analiza awaryjności sieci wodociągowej w aspekcie bezpieczeństwa funkcjonowania infrastruktury krytycznej, XV Konferencja Ochrony Środowiska, Chemik 68, 2014

[88] Szymczak M., Słownik języka polskiego, PWN, Warszawa 1993, s. 362.

[89] Tokarski J. (red.), Słownik wyrazów obcych, PWN, Warszawa 1980, s. 404.

[90] Tatjewski P., Sterowanie zaawansowane obiektów przemysłowych – struktura i algorytmy, wyd. Akademia Oficyna Wydawnicza EXIT, Warszawa 2002.

[91] Urbaniak A., Control algorithms of infiltration water intake unde uncertainty, 14th International Caprathian Control Conference (ICCC), 2013, s. 395-399.

[92] Urbaniak A., Monitorowanie i sterowanie procesami wodociągowymi, Wodociągi i Kanalizacja. Wyd. Komunalne, nr 7/8 , Poznań 2006 s. 36-39.

[93] Urbaniak A., Podstawy automatyki, Wyd. Politechniki Poznańskiej, (wyd. 3 poprawione) Poznań 2007.

[94] Urbaniak A., Sozański M., Informatyczne systemy w eksploatacji zakładów uzdatniania wody i oczyszczania ścieków, Gaz, Woda i Technika Sanitarna nr 9, 2001.

[95] Ustawa z dnia 7 czerwca 2001 r. o zbiorowym zaopatrzeniu w wodę i zbiorowym odprowadzaniu ścieków, źródło: https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20010720747.

[96] Ustawa z dnia 16 kwietnia 2004 roku o wyrobach budowlanych, źródło: https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=wdu20040920881.

[97] Water Distribution System Analysis Field Studies, Modeling and Management A Reference Guide for Utility, U. S. Environmental Procetion Agency, 2015.

[98] Wawrzynek J.: Metody opisu i wnioskowania statystycznego. Wrocław: Wyd. Akademii Ekonomicznej im. Oskara Langego we Wrocławiu, 2007, s.56–57. ISBN978-83-7011-859-4.

[99] Winczaszek M. Wybrane problemy szeregowania z optymalnym doborem przedziałów zakończenia wykonywania zadań. Rozprawa doktorska 2006 Politechnika Wrocławska

[100] Wind` G. H., Jean-Luc de Kok, Design and application of decision-support systems for integrated water management: lessons to be learnt, Physics and Chemistry of the Earth, vol. 28, 2003.

[101] Wodociągi i kanalizacja – Poradnik, Arkady 1971.

[102] Wolanin J., Zarys teorii bezpieczeństwa obywateli. Ochrona ludności na czas pokoju, DANMAR, Warszawa 2005, s. 54.

[103] Wyczółkowski R, Moczulski W. Concept of intelligent monitoring of local water supply system. Artificial Intelligence Methods AI-METH. Gliwice: Silesian University of Technology, 2005; 147-148.

[104] Wyczółkowski R., Przystałka P., Metoda rozmieszczania urządzeń pomiarowych i wykonawczych do celu diagnozowania wycieków w sieciach wodociągowych, Problemy eksploatacji, 2 -2011

[105] Wyczółkowski R., Paszkowski W., Loska A., Komoniewski M., Przegląd możliwości i potrzeb wspomagania zarządzania i monitorowania systemu wodociągowego w ujęciu idei Smart City, Innowacje w Zarządzaniu i Inżynierii Produkcji, Polskie Towarzystwo Inżynierii Produkcji, 2016, s. 570 – 583.

[106] Xia Li, Guo-jin Li, Leak Detection of Municipal Water Supply Network Based on the Cluster-Analysis and Fuzzy Pattern Recognition, IEEE Xplore, 2010, DOI: 10.1109/ICEEE.2010.5660550.

[107] Xu J.,Wang H., Rao J., Wang J., Zone scheduling optimization of pumps in water distribution networks with deep reinforcement learning and knowledge-assisted learning,Soft Computing, 2021 – Springer

[108] Xuan H., Yongming H., Bin Y., Zhiqiang G., Jinzhen F., Novel leakage detection and water loss management of urban water supply network using multiscale neural networks, Journal of Cleaner Production, 2021 Volume 278, Elsevier, 2021.

[109] Zakrzewski P., Mazurkiewicz J., Antonowicz A., Wykorzystanie algorytmów sztucznej inteligencji do dynamicznego określania punktu pracy obiektów wod-kan, Instal 2018 nr 9, 2018, s. 53-57.

[110] Zimoch I., Modele genetyczne w optymalizacji monitoringu sieci wodociągowej, INSTAL; Teoria i praktyka w instalacjach, ISSN 1640-8160, 2011, nr 7-8 s. 3-20.

[111] Źródło internetowe: http://www.instsani.pl/918/dobor-zbiornikow-na-wode (27.11.2021, g. 12:31).

[112] Źródło internetowe: http://www.instsani.pl/880/sieci-wodociagowe-2 (27.11.2021, g. 3:55)

[113] Źródło internetowe: https://www.cs.put.poznan.pl/pzakrzewski/sw/sw_wyk%C5%82ad_3.pdf (05.06.2022, g. 13:15).

[114] Źródło internetowe: Informacje na temat systemów SCADA, http://www.cs.put.poznan.pl/rklaus/scada/ (01.12.2021, g. 10:54).

[115] Źródło internetowe: https://www.vtscada.com/process-displays/ (01.12.2021, g. 09:20).

[116] Źródło internetowe: https://mfiles.pl/pl/index.php/Model (01.12.2021 g: 15:42).

[117] Źródło internetowe: Dokumentacja WNTR https://wntr.readthedocs.io/en/latest/index.html (05.12.2021, g. 08:44).

[118] Źródło internetowe (Kalibracja Opola): https://www.kierunekwodkan.pl/artykul,78430,gis-i-model-hydrauliczny-jako-nierozlaczne-narzedzia.html (23.11.2021, g. 18:44).

[119] Źródło internetowe: https://modelowaniesieci.pl/modelowanie-sieci-wodociagowej/ dostęp (05.12.2021, g. 14:12).

[120] Źródło internetowe: https://www.bentley.com/pl/about-us/news/2018/february/06/understanding-calibration-of-water-distribution-models (05.12.2021, g. 14:20).

[121] Źródło internetowe https://mfiles.pl/pl/index.php/Klasyfikacja definicja klasyfikacji (14.12.2021, g. 23:15).

[122] Źródło internetowe http://mariusz.uchronski.staff.iiar.pwr.wroc.pl/TI/TI_zadanie_3_rozw.pdf, (15.01.2022, g. 12:54).

[123] Źródło internetowe https://www.automatyka.pl/produkty/tp-aqua-zintegrowany-system-informatyczny-dla-branzy-wod-kan-15844-2, (01.07.2022, g. 12:40).

[124] Źródło internetowe https://edams.com/industries/water-sanitation/, (02.07.2022, g. 23:15).

Spis rysunków

Rys. 2.1 Ogólna struktura systemów sterowania [2, 90] 23
Rys. 2.2 Warstwowa struktura systemów sterowania [2, 3, 90, 113] 24
Rys. 2.3 Struktura systemu eksperckiego [3, 57] 28
Rys. 2.4 Trójpoziomowa architektura systemów informatycznych [15] 32
Rys. 2.5 Przykładowy ekran synoptyczny systemu SCADA [115] 33
Rys. 2.6 Dwuwarstwowa struktura sterowania siecią dystrybucji wody [52] 37
Rys. 3.1 Topologia sieci rozgałęzieniowej [19, 112] 46
Rys. 3.2 Topologia sieci obwodowej [19, 112] 47
Rys. 3.3 Topologia sieci mieszanej [19, 112] 48
Rys. 3.4 Schemat pojęciowy własności systemu [70] 51
Rys. 3.5 Istota analizy i oceny ryzyka w systemach dystrybucji wody [70] 53
Rys. 4.1 Schemat blokowy procesu modelowania matematycznego [42] 56
Rys. 4.2 Interfejs programu EPANET [opracowanie własne] 58
Rys. 4.3 Graf przykładowych przypadków użycia WNTR [117] 60
Rys. 4.4 Wykres rozbioru w modelowanym systemie dystrybucji [opracowanie własne] 64
Rys. 4.5 Pasek narzędzi programu EPANET [opracowanie własne] 69
Rys. 4.6 Modelowanie rezerwuaru i zbiornika w programie EPANET [opracowanie własne] 70
Rys. 4.7 Modelowanie węzła poboru oraz przewodu w programie EPANET [opracowanie własne] 71
Rys. 4.8 Modelowanie pompy oraz zasuwy w programie EPANET [opracowanie własne] 72
Rys. 4.9 Wynik modelowania przykładowej sieci w programie EPANET [opracowanie własne] 73
Rys. 4.10 Przykład wzorca rozbioru w programie EPANET [opracowanie własne] 73
Rys. 4.11 Przykład wprowadzania krzywej w programie EPANET [opracowanie własne] 74
Rys. 4.12 Struktura bazy danych przechowującej informacje o sieciach wodociągowych [opracowanie własne] 80
Rys. 4.13 Modelowanie awarii (wycieku) z wykorzystaniem środowiska EPANET [opracowanie własne] 82
Rys. 4.14 Modelowanie awarii (pęknięcia) z wykorzystaniem środowiska EPANET [opracowanie własne] 84
Rys. 5.1 Sposób interpretacji zasuw odcinających w środowisku EPANET [opracowanie własne] 88
Rys. 5.2 Przykładowy model sieci wodociągowej z lokalizacją zasuw odcinających [opracowanie własne] 90
Rys. 5.3 Przykładowy segment (S15) w analizowanym przypadku sieci wodociągowej [opracowanie własne] 98
Rys. 5.4 Wykres przepływów w przewodzie P38 w przeprowadzonych symulacjach [opracowanie własne] 101
Rys. 7.1 Przykład reprezentacji graficznej działania algorytmu klasyfikacji elementów sieci wodociągowej [opracowanie własne] 120
Rys. 7.2 Schemat przepływu danych *Algorytmu 1* [opracowanie własne] 121

Rys. 7.3 Przykład reprezentacji graficznej działania algorytmu wyznaczania trasy dla pierwszej godziny symulacji [opracowanie własne] 124
Rys. 7.4 Przykład reprezentacji graficznej działania algorytmu wyznaczania trasy dla siódmej godziny symulacji [opracowanie własne] 124
Rys. 7.5 Schemat przepływu danych *Algorytmu 2* [opracowanie własne] 125
Rys. 7.6 Wynik działania algorytmu wyznaczającego segmenty w analizowanym systemie dystrybucji wody [opracowanie własne] 127
Rys. 7.7 Schemat przepływu danych *Algorytmu 3* [opracowanie własne] 128
Rys. 7.8 Schemat przepływu danych *Algorytmu 4* [opracowanie własne] 134
Rys. 7.9 Przykładowy przypadek testowy [opracowanie własne] 136
Rys. 7.10 Trasy przepływów wody do węzłów krytycznych wraz z lokalizacją awarii [opracowanie własne] 138
Rys. 7.11 Schemat przepływu danych *Algorytmu 5* [opracowanie własne] 139
Rys. 7.12 Wynik działania Algorytmu 6 – agregacja awarii [opracowanie własne] 142
Rys. 7.13 Schemat przepływu danych *Algorytmu 6* [opracowanie własne] 143
Rys. 7.14 Rezultat wywołania metody *create_scenario* [opracowanie własne] 150
Rys. 7.15 Rezultat wywołania metody *graph_with_weights* [opracowanie własne] ... 151
Rys. 7.16 Trasy poszczególnych ekip naprawczych (Algorytm 7) [opracowanie własne] 153
Rys. 7.17 Schemat przepływu danych *Algorytmu 7* [opracowanie własne] 155
Rys. 8.1 Schemat integracji narzędzi tworzących środowisko symulacyjne [opracowanie własne] 158
Rys. 8.2 Topologia sieci [opracowanie własne] 159
Rys. 8.3 Topologia sieci Net3 z lokalizacją zasuw i sektorów [opracowanie własne] 160
Rys. 8.4 Topologia sieci Środy Wielkopolskiej [opracowanie własne] 162
Rys. 8.5 Topologia sieci Środy Wielkopolskiej z lokalizacją zasuw i sektorów [opracowanie własne] 163
Rys. 8.6 Scenariusz testowy Net3 – Scenario_1 [opracowanie własne] 165
Rys. 8.7 Rozwiązanie problemu szeregowania zadań – Scenario_1 (z agregacją) [opracowanie własne] 166
Rys. 8.8 Rozwiązanie problemu szeregowania zadań – Scenario_1 (bez agregacji) [opracowanie własne] 167
Rys. 8.9 Scenariusz testowy Net3 – Scenario_2 [opracowanie własne] 168
Rys. 8.10 Rozwiązanie problemu szeregowania zadań – Scenario_2 (z agregacją) [opracowanie własne] 169
Rys. 8.11 Rozwiązanie problemu szeregowania zadań – Scenario_2 (bez agregacji) [opracowanie własne] 170
Rys. 8.12 Scenariusz testowy Środa Wlkp. – Scenario_1 [opracowanie własne] 172
Rys. 8.13 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_1 (z agregacją) [opracowanie własne] 173
Rys. 8.14 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_1 (bez agregacji) [opracowanie własne] 174
Rys. 8.15 Scenariusz testowy Środa Wlkp. – Scenario_2 [opracowanie własne] 176
Rys. 8.16 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_2 (z agregacją) [opracowanie własne] 178
Rys. 8.17 Rozwiązanie problemu szeregowania zadań – Środa Wlkp. Scenario_2 (bez agregacji) [opracowanie własne] 179

Spis tabel

Tab. 4.1 *Informacje o zbiornikach w modelowanym systemie [opracowanie własne]* . 62
Tab. 4.2 *Informacje o przykładowych węzłach w modelowanym systemie [opracowanie własne]* 63
Tab. 4.3 *Informacje o wzorcach rozbioru w modelowanym systemie [opracowanie własne]* 65
Tab. 4.4 *Informacje o przewodach w modelowanym systemie [opracowanie własne]* . 66
Tab. 4.5 *Informacje o pompach w modelowanym systemie [opracowanie własne]*..... 67
Tab. 4.6 *Informacje o krzywych w modelowanym systemie [opracowanie własne]* 68
Tab. 5.1 *Informacje o zasuwach odcinających w modelowanym systemie [opracowanie własne]* 91
Tab. 5.2 (M*acierz A) analizowanego przykładu SDW [opracowanie własne]*..... 92
Tab. 5.3 (M*acierz B) analizowanego przykładu SDW [opracowanie własne]*..... 93
Tab. 5.4 (M*acierz C) analizowanego przykładu SDW [opracowanie własne]* 94
Tab. 5.5 *Prezentacja działania pierwszego etapu algorytmu (Macierz C) [opracowanie własne]* 96
Tab. 5.6 Prezentacja działania drugiego etapu algorytmu (Macierz A) *[opracowanie własne]* 97
Tab. 6.1 *Średnie czasy trwania przykładowych czynności naprawczych [10]* 106
Tab. 7.1 *Przykład wyników symulacji Algorytmu 1 [opracowanie własne]*..... 118
Tab. 7.2 *Określenie przewidywanego czasu naprawy w zależności od typu awarii [10]* 129
Tab. 7.3 *Przykład awarii oraz ich klasyfikacja w przykładowym modelu testowym [opracowanie własne]* 137
Tab. 7.4 *Przykład awarii oraz ich klasyfikacja w przykładowym modelu testowym [opracowanie własne]* 141
Tab. 7.5 *Wyniki utworzenia poziomów awarii dla opisywanego przykładu [opracowanie własne]* 145
Tab. 7.6 *Wymagania materiałowo-sprzętowe w celu naprawy awarii [opracowanie własne]* 148
Tab. 7.7 *Wykres Gantta dla analizowanego przykładu działania Algorytmu 7A [opracowanie własne]* 154
Tab. 8.1 *Informacje o scenariuszach testowych [opracowanie własne]* 164
Tab. 8.2 *Informacje o scenariuszach testowych – informacje podstawowe [opracowanie własne]* 182
Tab. 8.3 *Informacje o scenariuszach testowych – zużycie materiałów [opracowanie własne]* 183

Załączniki

Załącznik A – Kod programu analizującego przepływy wraz z wynikami przykładowego scenariusza testowego

```
import wntr
import wntr.network.controls as controls


class FlowBalance:

    def __init__(self, scenario_file_path, valve_file_path, inp_file_path=None):

        self.scenario_file_path = scenario_file_path
        self.scenario_data = open(scenario_file_path + '.txt', 'r+')
        self.valve_data = open(valve_file_path + '.txt', 'r')

        self.scenario_cases = dict()
        self.scenario_file_converter()      # Convert the scenario file to list of dict

        self.valve_id_list = list()
        self.valve_file_converter()         # Convert the valve file to list of dict

        self.inp_file_path = inp_file_path
        self.wn = wntr.network.WaterNetworkModel('networks/' + self.inp_file_path + '.inp')
        self.sim = wntr.sim.WNTRSimulator(wn=self.wn)

        self.flow_balance_algorithm_result_list = list()

    def scenario_file_converter(self):
        analyzed_example_list = list()
        for line in self.scenario_data:
            if line[0] == '#':
                split_line = line.split('|')
                self.scenario_cases['scenario_id'] = split_line[1]
            elif line[0:3] == '(P)':
                self.scenario_cases['analyzed_pipe'] = line[4:-1]
            else:
                split_valve_list = line[4:-1].split(',')
                analyzed_example_list.append(split_valve_list)
        self.scenario_cases['valve_to_close'] = analyzed_example_list

    def valve_file_converter(self):
        for line in self.valve_data:
            if line[0] != "#":
                self.valve_id_list.append(line[:-1].split(','))

    def flow_balance(self, analyzed_pipe, use_case, sim_time=24):

        pipe_flow_result = list()
        flow_result = list()
        closed_pipe = list()
        sum_of_flowrate = 0

        self.wn = wntr.network.WaterNetworkModel('networks/' + self.inp_file_path + '.inp')
        self.sim = wntr.sim.WNTRSimulator(wn=self.wn)

        for valve in use_case:
            for valve_id in self.valve_id_list:
                if valve == valve_id[0] and valve_id[2] != analyzed_pipe:
                    closed_pipe.append(valve_id[2])

        for pipe in closed_pipe:

            act = controls.ControlAction(self.wn.get_link(pipe), 'status', 0)
            cond = controls.SimTimeCondition(self.wn,
                                             controls.Comparison.ge, '00:00:00')
            ctr1 = controls.Control(cond, act)
            self.wn.add_control('control' + str(pipe), ctr1)
```

```
        for step in range(0, sim_time + 1, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()
            flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

        for x in range(0, len(flow_result), 1):   # LPS to GAL/MIN
            flow_result[x][x * 3600][analyzed_pipe] = round(flow_result[x]
                                        [x * 3600][analyzed_pipe] * 15.85, 2)
            sum_of_flowrate = sum_of_flowrate + round(flow_result[x]
                                        [x * 3600][analyzed_pipe] * 15.85, 2)
            pipe_flow_result.append(flow_result[x][x * 3600][analyzed_pipe])

        return {'analyzed_pipe': analyzed_pipe, 'analyzed_use_case': use_case,
                  'closed_pipe': closed_pipe, 'sum_of_flowrate':
                  round(sum_of_flowrate, 2), 'flow_result': pipe_flow_result}

    def flow_balance_algorithm(self):

        next_scenario = list()
        self.flow_balance_algorithm_result_list = list()

        for case in self.scenario_cases['valve_to_close']:
            result = self.flow_balance(self.scenario_cases['analyzed_pipe'], case)

            with open(self.scenario_file_path + '_report.txt', "a") as fp:
                print(result, file=fp)

            if result['sum_of_flowrate'] <= 0.0:
                self.flow_balance_algorithm_result_list.append('T')
                next_scenario.append(result['analyzed_use_case'])
            else:
                self.flow_balance_algorithm_result_list.append('F')

        if len(next_scenario) >= 2:
            pass

        self.write_results()

    def write_results(self):

        with open(self.scenario_file_path + '.txt') as fp:
            lines = fp.read().splitlines()
        with open(self.scenario_file_path + '_result.txt', "w") as fp:
            print(lines[0], file=fp)
            print(lines[1], file=fp)
            for x in range(2, len(lines), 1):
                print(lines[x] + ' ' + str(self.flow_balance_algorithm_result_list
                                                                [x-2]), file=fp)
    def day_flow(self, pipe_to_check, sim_time = 24):

        sum_of_flow = 0
        pipe_flow = 0
        flow_result = list()

        self.wn = wntr.network.WaterNetworkModel('networks/' + self.inp_file_path + '.inp')
        self.sim = wntr.sim.WNTRSimulator(wn=self.wn)

        act = controls.ControlAction(self.wn.get_link('P11'), 'status', 0)
        cond = controls.SimTimeCondition(self.wn, controls.Comparison.ge, '00:00:00')
        ctr1 = controls.Control(cond, act)
        self.wn.add_control('control' + str('P11'), ctr1)

        for step in range(0, sim_time + 1, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()
            flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

        for step in range(0, len(flow_result), 1):

            for key in flow_result[step][step*3600]:
                flow_result[step][step * 3600][key] =
                  round(flow_result[step][step*3600][key] * 15.85, 2)
                sum_of_flow = sum_of_flow + flow_result[step][step * 3600][key]
```

```
                if key == pipe_to_check:
                    pipe_flow = pipe_flow + flow_result[step][step * 3600][key]

        print(f'Sum of flow: {round(sum_of_flow, 2)}')
        print(f'Pipe flow: {round(pipe_flow, 2)}')

        print(f'%: {round((round(pipe_flow, 2) * 100) / round(sum_of_flow, 2), 2)}')

new_case = FlowBalance('scenario', 'valve_list', 'ariel_phd')
# new_case.flow_balance_algorithm()
# new_case.day_flow(pipe_to_check='P1')
new_case.total_length_pipe('P38')
```

Zawartość pliku *scenario.txt*:

```
#==== Scenario stage (1) | Net: AA_pdh.inp | Use case: 1
(P) P38
(1) V20,V26,V27,V28
(2) V20,V26,V27,V31
(3) V20,V26,V28,V31
(4) V20,V27,V28,V31
(5) V26,V27,V28,V31
```

Zawartość pliku *scenario_result.txt*:

```
#==== Scenario stage (1) | Net: AA_pdh.inp | Use case: 1
(P) P38
(1) V20,V26,V27,V28 T
(2) V20,V26,V27,V31 F
(3) V20,V26,V28,V31 F
(4) V20,V27,V28,V31 F
(5) V26,V27,V28,V31 F
```

Zawartość pliku *scenario_report.txt*:

```
{'analyzed_pipe': 'P38',
 'analyzed_use_case': ['V20', 'V26', 'V27', 'V28'],
 'closed_pipe': ['P28', 'P32', 'P37', 'P30'],
 'sum_of_flowrate': -27.41,
  flow_result': [-0.01, -0.01, -0.02, -0.02, -0.03, -0.05, -0.05,
                 -0.07, -0.08, -0.08, -0.1, -0.1, -0.12, -0.12,
                 -0.1, -0.1, -0.09, -0.11, -0.1, -0.09, -0.09,
                 -0.08, -0.06, -0.04, -0.01]}

{'analyzed_pipe': 'P38',
 'analyzed_use_case': ['V20', 'V26', 'V27', 'V31'],
 'closed_pipe': ['P28', 'P32', 'P37'],
 'sum_of_flowrate': 25.69,
 'flow_result': [0.01, 0.01, 0.02, 0.02, 0.03, 0.04, 0.04, 0.07,
                 0.07, 0.07, 0.09, 0.09, 0.11, 0.11, 0.09, 0.09,
                 0.08, 0.11, 0.1, 0.09, 0.09, 0.08, 0.06, 0.04,
                 0.01]}

{'analyzed_pipe': 'P38',
 'analyzed_use_case': ['V20', 'V26', 'V28', 'V31'],
 'closed_pipe': ['P28', 'P32', 'P30'],
 'sum_of_flowrate': 10.0,
 'flow_result': [0.0, 0.0, 0.01, 0.01, 0.01, 0.02, 0.02, 0.03,
                 0.03, 0.03, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04,
                 0.03, 0.04, 0.04, 0.03, 0.03, 0.03, 0.02, 0.01,
                 0.0]}
```

```
{'analyzed_pipe': 'P38',
'analyzed_use_case': ['V20', 'V27', 'V28', 'V31'],
'closed_pipe': ['P28', 'P37', 'P30'],
'sum_of_flowrate': 23.16,
'flow_result': [0.01, 0.01, 0.02, 0.02, 0.03, 0.04, 0.04, 0.06,
              0.07, 0.07, 0.08, 0.08, 0.1, 0.1, 0.08, 0.08,
                0.07, 0.09, 0.09, 0.08, 0.08, 0.07, 0.05, 0.03,
                0.01]}

{'analyzed_pipe': 'P38',
'analyzed_use_case': ['V26', 'V27', 'V28', 'V31'],
'closed_pipe': ['P32', 'P37', 'P30'],
'sum_of_flowrate': 35.02, 'flow_result': [0.01, 0.01, 0.03, 0.03, 0.04, 0.06, 0.06,
0.09,
                0.1, 0.1, 0.13, 0.13, 0.15, 0.15, 0.13, 0.13,
                0.11, 0.14, 0.13, 0.12, 0.12, 0.1, 0.08, 0.05,
                0.01]}
```

Załącznik B – Plik konfiguracyjny (config.py)

```
class Configuration_data:
    def __init__(self):

        # ========================================================= MODEL AND SIMULATION

        self.inp_file_name = 'Sroda_bez_brodowa'
        self.sim_duration = 24

        # ============================================================ PATHS

        self.networks_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\networks\\'
        self.additional_data_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\
                                     networks\\additional_data\\'
        self.results_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\results\\'
        self.valve_info_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\valves\\'
        self.segments_info_path = 'C:\\Users\\ariel\\PycharmProjects\\
                                   doctorate\\segments\\'
        self.matrix_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\matrix\\'
        self.scenarios_path = 'C:\\Users\\ariel\\PycharmProjects
                               \\doctorate\\scenarios\\'
        self.failures_path = 'C:\\Users\\ariel\\PycharmProjects\\doctorate\\failures\\'

        # ========================================================== ALGORITHM 1

        self.nodes_csv_file = self.inp_file_name + '_nodes'
        self.links_csv_file = self.inp_file_name + '_links'

        self.flow_category = {'CAT-1': [90.0, float('inf')],
                              'CAT-2': [75.0, 89.9],
                              'CAT-3': [50.0, 74.9],
                              'CAT-4': [float('-inf'), 49.9]}

        self.head_category = {'CAT-1': [90.0, float('inf')],
                              'CAT-2': [75.0, 89.9],
                              'CAT-3': [50.0, 74.9],
                              'CAT-4': [float('-inf'), 49.9]}

        self.demand_category = {'CAT-1': [90.0, float('inf')],
                                'CAT-2': [75.0, 89.9],
                                'CAT-3': [50.0, 74.9],
                                'CAT-4': [float('-inf'), 49.9]}

        self.pressure_category = {'CAT-1': [90.0, float('inf')],
                                  'CAT-2': [75.0, 89.9],
                                  'CAT-3': [50.0, 74.9],
                                  'CAT-4': [float('-inf'), 49.9]}

        # ========================================================== ALGORITHM 3
        self.valve_info = self.inp_file_name + '_valve_info'
        self.valve_set = 'set_1'
        self.segments = self.inp_file_name + '_segments'

        self.matrix_a = self.inp_file_name + '_matrixA'
        self.matrix_b = self.inp_file_name + '_matrixB'
        self.matrix_c = self.inp_file_name + '_matrixC'

        # ========================================================== ALGORITHM 4
        self.failures_info = self.inp_file_name + '_failures_info'
        self.failures_set = 'set_2'

        self.historical_data_random = [3, 10]

        multiplier = 0.0254 # in to m
        self.map_pipe_diameter ={'D3': [0, 14*multiplier],
                                 'D2': [14*multiplier, 18*multiplier],
                                 'D1': [18*multiplier, 24*multiplier]}
```

```
        self.repair_times = {
            'D3': {
                'open_valve': [10, 15],
                'close_valve': [10, 15],
                'failure_localisation': [20, 25],
                'asphalt_extraction': [70, 120],
                'removal_of_paving_slabs': [30, 35],
                'excavation_for_the_pipeline': [80, 100],
                'pumping_out_the_excavation': [30, 60],
                'cleaning_the_pipe': [30, 35],
                'mount_the_bound': [25, 30],
                'assembly_of_the_seal': [40, 45],
                'pipe_remove': [30, 40],
                'add_new_pipe': [50, 60],
                'disinfection': [20, 25],
                'backfilling_the_conduit': [40, 50],
                'finishing_and_cleaning_activities': [60, 80]
            },
            'D2': {
                'open_valve': [15, 20],
                'close_valve': [15, 20],
                'failure_localisation': [55, 60],
                'asphalt_extraction': [120, 160],
                'removal_of_paving_slabs': [45, 50],
                'excavation_for_the_pipeline': [100, 140],
                'pumping_out_the_excavation': [80, 120],
                'cleaning_the_pipe': [40, 45],
                'mount_the_bound': [40, 50],
                'assembly_of_the_seal': [50, 55],
                'pipe_remove': [50, 60],
                'add_new_pipe': [60, 70],
                'disinfection': [30, 35],
                'backfilling_the_conduit': [55, 65],
                'finishing_and_cleaning_activities': [75, 100]
            },
            'D1': {
                'open_valve': [20, 30],
                'close_valve': [20, 30],
                'failure_localisation': [50, 60],
                'asphalt_extraction': [160, 240],
                'removal_of_paving_slabs': [60, 65],
                'excavation_for_the_pipeline': [150, 240],
                'pumping_out_the_excavation': [120, 180],
                'cleaning_the_pipe': [50, 60],
                'mount_the_bound': [60, 90],
                'assembly_of_the_seal': [60, 90],
                'pipe_remove': [90, 140],
                'add_new_pipe': [180, 240],
                'disinfection': [50, 60],
                'backfilling_the_conduit': [70, 80],
                'finishing_and_cleaning_activities': [100, 160]
            }}

        self.failure_type = {
            'F1': ['failure_localisation', 'asphalt_extraction',
                   'removal_of_paving_slabs', 'excavation_for_the_pipeline',
                   'pumping_out_the_excavation',
                   'mount_the_bound', 'assembly_of_the_seal',
                   'backfilling_the_conduit', 'finishing_and_cleaning_activities'],
            'F2': ['open_valve', 'close_valve', 'failure_localisation',
                   'asphalt_extraction', 'removal_of_paving_slabs',
                   'excavation_for_the_pipeline', 'pumping_out_the_excavation',
                   'cleaning_the_pipe', 'assembly_of_the_seal', 'mount_the_bound',
                   'disinfection', 'backfilling_the_conduit',
                   'finishing_and_cleaning_activities'],
            'F3': ['open_valve', 'close_valve', 'failure_localisation',
                   'asphalt_extraction', 'removal_of_paving_slabs',
                   'excavation_for_the_pipeline', 'pumping_out_the_excavation',
                   'cleaning_the_pipe', 'assembly_of_the_seal', 'pipe_remove',
                   'add_new_pipe', 'disinfection', 'backfilling_the_conduit',
                   'finishing_and_cleaning_activities']}
```

```
        # ========================================================= ALGORITHM 5
        self.additional_info = self.inp_file_name + '_additional_info'
        self.additional_set = 'set_1'

        # ========================================================= ALGORITHM 6
        self.aggregation_result = self.inp_file_name + '_aggregation'
        self.aggregation_set = 'set_1'

        # ========================================================= ALGORITHM 7
        self.scenario_file = self.inp_file_name + '_scenario'
        self.scenario_set = 'set_1'

        self.repair_cost = {'F1': {'clamps': 2, 'pipes': 0, 'other_stuff': 10},
                            'F2': {'clamps': 4, 'pipes': 0, 'other_stuff': 15},
                            'F3': {'clamps': 0, 'pipes': 2, 'other_stuff': 20}}
```

Załącznik C – Kod źródłowy Algorytmu 1

```
"""
          Name:     Algorithm 1 - Klasyfikacja elementów sieci wodociągowej
        Author:     Ariel Antonowicz
   Last update:     24.02.2022
"""

import wntr
import copy
import pandas as pd
import os, glob
from config import Configuration_data


class Algorithm_1:

    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'
        self.nodes_csv_file = self.init_config.results_path + self.init_config.nodes_csv_file +
                                                                                                '.csv'
        self.links_csv_file = self.init_config.results_path + self.init_config.links_csv_file +
                                                                                                '.csv'

        self.link_file = open(file=self.links_csv_file, newline='', mode='a')
        self.node_file = open(file=self.nodes_csv_file, newline='', mode='a')
        self.sum_of_flow = 0
        self.sum_of_pressure = 0
        self.sum_of_demand = 0
        self.sum_of_head = 0
        self.links_results = {'flow': list()}
        self.nodes_results = {'head': list(), 'demand': list(), 'pressure': list()}
        self.isolated_nodes_list = list()
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def run_simulation(self, wn):
        simulation_results = list()
        sim = wntr.sim.EpanetSimulator(wn=wn)
        for step in range(0, self.init_config.sim_duration + 1, 1):
            wn.options.time.duration = step * 3600
            simulation_results = sim.run_sim()

        return {'flow': simulation_results.link['flowrate'].to_dict('index'),
                'head': simulation_results.node['head'].to_dict('index'),
                'demand': simulation_results.node['demand'].to_dict('index'),
                'pressure': simulation_results.node['pressure'].to_dict('index')}

    def create_new_topology(self, element_type, element_id):
        wn = wntr.network.WaterNetworkModel(self.inp_file)

        if element_type == 'node':
            # Search Pipe ID where element is start or end node.
            for link in wn.link_name_list:
                if element_id == str(wn.get_link(link).todict()['start_node']):
                    # Copy info about NODE and LINK
                    node_temp_info = copy.deepcopy(wn.get_node(element_id).todict())
                    link_temp_info = copy.deepcopy(wn.get_link(link).todict())
                    wn.remove_link(wn.get_link(link).todict()['name'])  # Remove link
                    wn.remove_node(element_id)  # Remove analyzed element

                    # Create new node and link base on temp_info
                    wn.add_junction(name=element_id,
                                    base_demand=0,
                                    elevation=wn.get_node(link_temp_info['end_node'])
                                                                            .todict()['elevation'],
                                    coordinates=node_temp_info['coordinates'])
                    wn.add_pipe(name=link_temp_info['name'],
                                start_node_name=element_id,
                                end_node_name=link_temp_info['end_node_name'])
                    break
```

```
                elif element_id == str(wn.get_link(link).todict()['end_node']):
                    # Copy info about NODE and LINK
                    node_temp_info = copy.deepcopy(wn.get_node(element_id).todict())
                    link_temp_info = copy.deepcopy(wn.get_link(link).todict())
                    wn.remove_link(wn.get_link(link).todict()['name'])  # Remove link
                    wn.remove_node(element_id)  # Remove analyzed element
                    # Create new temporary junction
                    wn.add_junction(name=element_id,
                                    base_demand=0,
                                    elevation=wn.get_node(link_temp_info['start_node'])
                                                                        .todict()['elevation'],
                                    coordinates=node_temp_info['coordinates'])

                    wn.add_pipe(name=link_temp_info['name'],  # Create new pipe
                                start_node_name=link_temp_info['start_node_name'],
                                end_node_name=element_id)
                    break
        elif element_type == 'link':
            pump_temp_info = copy.deepcopy(wn.get_link(element_id).todict())
            wn.remove_link(element_id)
            wn.add_pipe(name=pump_temp_info['name'],
                        start_node_name=pump_temp_info['start_node_name'],
                        end_node_name=pump_temp_info['end_node_name'])

        elif element_type == 'pipe':
            not_isolated_node_list = list()

            # Check that remove pipe creates isolated junction
            wn.remove_link(element_id)  # Remove analyzed element

            for link in wn.link_name_list:
                not_isolated_node_list.append(wn.get_link(link).todict()['start_node_name'])
                    if wn.get_link(link).todict()['start_node_name'] not in not_isolated_node_list
                    else not_isolated_node_list
                not_isolated_node_list.append(wn.get_link(link).todict()['end_node_name'])
                    if wn.get_link(link).todict()['end_node_name'] not in not_isolated_node_list
                    else not_isolated_node_list

            self.isolated_nodes_list = list(set(not_isolated_node_list)
                                                .symmetric_difference(set(wn.node_name_list
            for x in self.isolated_nodes_list:
                wn.remove_node(x)

        return wn

    def reference_simulation(self):
        # Create reference result list
        # print(f'Start reference simulation for: {self.inp_file}')
        wn = wntr.network.WaterNetworkModel(self.inp_file)
        sim_result = self.run_simulation(wn)
        reference_simulation_result = {
            'ID': 'Reference sim',
            'isolated_nodes_id': [],
            'flow': sim_result['flow'],
            'head': sim_result['head'],
            'demand': sim_result['demand'],
            'pressure': sim_result['pressure']}

        return reference_simulation_result

    def reservoir_resilience_simulation(self, list_to_test=None):
        if not list_to_test:
            list_to_check = list(self.wn.nodes.reservoir_names)
        else:
            list_to_check = list_to_test

        reservoir_resilience_simulation_result = list()
        for element in list_to_check:
            wn = self.create_new_topology('node', element)
            sim_result = self.run_simulation(wn)

            use_case = {
                'ID': element,
                'type': 'reservoir',
                'flow': sim_result['flow'],
                'head': sim_result['head'],
                'demand': sim_result['demand'],
                'pressure': sim_result['pressure'],
                'isolated_nodes_id': []
            }
            reservoir_resilience_simulation_result.append(use_case)
        return reservoir_resilience_simulation_result
```

```
    def tank_resilience_simulation(self, list_to_test=None):
        if not list_to_test:
            list_to_check = list(self.wn.nodes.tank_names)
        else:
            list_to_check = list_to_test
        tank_resilience_simulation_result = list()
        for element in list_to_check:
            wn = self.create_new_topology('node', element)
            sim_result = self.run_simulation(wn)
            use_case = {
                'ID': element,
                'type': 'tank',
                'flow': sim_result['flow'],
                'head': sim_result['head'],
                'demand': sim_result['demand'],
                'pressure': sim_result['pressure'],
                'isolated_nodes_id': []
            }
            tank_resilience_simulation_result.append(use_case)
        return tank_resilience_simulation_result

    def pump_resilience_simulation(self, list_to_test=None):
        if not list_to_test:
            list_to_check = list(self.wn.links.pump_names)
        else:
            list_to_check = list_to_test
        pump_resilience_simulation_result = list()
        for element in list_to_check:
            wn = self.create_new_topology('link', element)
            sim_result = self.run_simulation(wn)
            use_case = {
                'ID': element,
                'type': 'pump',
                'flow': sim_result['flow'],
                'head': sim_result['head'],
                'demand': sim_result['demand'],
                'pressure': sim_result['pressure'],
                'isolated_nodes_id': []
            }
            pump_resilience_simulation_result.append(use_case)
        return pump_resilience_simulation_result

    def valve_resilience_simulation(self, list_to_test=None):
        if not list_to_test:
            list_to_check = list(self.wn.links.valve_names)
        else:
            list_to_check = list_to_test
        valve_resilience_simulation_result = list()
        for element in list_to_check:
            wn = self.create_new_topology('link', element)
            sim_result = self.run_simulation(wn)
            use_case = {
                'ID': element,
                'type': 'valve',
                'flow': sim_result['flow'],
                'head': sim_result['head'],
                'demand': sim_result['demand'],
                'pressure': sim_result['pressure'],
                'isolated_nodes_id': []
            }
            valve_resilience_simulation_result.append(use_case)
        return valve_resilience_simulation_result

    def pipe_resilience_simulation(self, list_to_test=None):
        if not list_to_test:
            list_to_check = list(self.wn.links.pipe_names)
        else:
            list_to_check = list_to_test
        pipe_resilience_simulation_result = list()
        for element in list_to_check:
            wn = self.create_new_topology('pipe', element)
            sim_result = self.run_simulation(wn)
            use_case = {
                'ID': element,
                'type': 'pipe',
                'flow': sim_result['flow'],
                'head': sim_result['head'],
                'demand': sim_result['demand'],
                'pressure': sim_result['pressure'],
                'isolated_nodes_id': self.isolated_nodes_list
            }
            pipe_resilience_simulation_result.append(use_case)
        return pipe_resilience_simulation_result
```

```
    def create_data_frame_for_links(self, data_type, data):
        flow_data = dict()

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)

        for link_id in self.wn.link_name_list:
            flow_data[link_id] = 0

        if data_type == 'reference_simulation':
            for sim_step in data['flow']:
                for link_id in data['flow'][sim_step]:
                    flow_data[link_id] = flow_data[link_id] + data['flow'][sim_step][link_id]

            for element in flow_data.values():
                self.sum_of_flow = self.sum_of_flow + element
            flow_data['sum'] = self.sum_of_flow
            flow_data['percent'] = 100
            flow_data['category'] = 'CAT-1'
            flow_df = pd.DataFrame(flow_data, index=['Flow (ref_sim)'])
            flow_df.to_csv(self.link_file, index=True, header=True)

        elif data_type == 'resilience_simulation':
            index = list()
            temp_list = list()
            for case in data:

                flow_data.clear()
                for link_id in self.wn.link_name_list:
                    flow_data[link_id] = 0
                index.append('Flow (' + str(case['ID']) + ')')
                for sim_step in case['flow']:
                    for link_id in case['flow'][sim_step]:
                        flow_data[link_id] = flow_data[link_id] + case['flow'][sim_step][link_id]
                total_sum = 0
                for element in flow_data.values():
                    total_sum = total_sum + element
                flow_data['sum'] = total_sum
                flow_data['percent'] = round((total_sum * 100)/self.sum_of_flow , 2)

                flow_data['remove_element_id'] = case['ID']
                flow_data['element_type'] = case['type']

                for category in self.init_config.flow_category:
                    if self.init_config.flow_category[category][0] <= flow_data['percent'] <=
                                                self.init_config.flow_category[category][1]:
                        flow_data['category'] = category
                        break
                temp_list.append(copy.deepcopy(flow_data))

                self.links_results['flow'].append(copy.deepcopy(flow_data))

            flow_df = pd.DataFrame(temp_list, index=index)
            flow_df.to_csv(self.link_file, index=True, header=False)

        return self.links_results

    def create_data_frame_for_nodes(self, data_type, data):
        head_data = dict()
        demand_data = dict()
        pressure_data = dict()
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)

        for node_id in self.wn.node_name_list:
            head_data[node_id] = 0
            demand_data[node_id] = 0
            pressure_data[node_id] = 0

        if data_type == 'reference_simulation':
            for sim_step in data['head']:
                for node_id in data['head'][sim_step]:
                    head_data[node_id] = head_data[node_id] + data['head'][sim_step][node_id]
            for sim_step in data['demand']:
                for node_id in data['demand'][sim_step]:
                    demand_data[node_id] = demand_data[node_id] + data['demand'][sim_step][node_id]
            for sim_step in data['pressure']:
                for node_id in data['pressure'][sim_step]:
                    pressure_data[node_id] = pressure_data[node_id] +
                                                    data['pressure'][sim_step][node_id]

            for element in head_data.values():
                self.sum_of_head = self.sum_of_head + element
            head_data['sum'] = self.sum_of_head
            head_data['percent'] = 100
            head_data['category'] = 'CAT-1'
```

```
            for element in demand_data.values():
                self.sum_of_demand = self.sum_of_demand + element
            demand_data['sum'] = self.sum_of_demand
            demand_data['percent'] = 100
            demand_data['category'] = 'CAT-1'

            for element in pressure_data.values():
                self.sum_of_pressure = self.sum_of_pressure + element
            pressure_data['sum'] = self.sum_of_pressure
            pressure_data['percent'] = 100
            pressure_data['category'] = 'CAT-1'

            head_df = pd.DataFrame(head_data, index=['Head (ref_sim)'])
            demand_df = pd.DataFrame(demand_data, index=['Demand (ref_sim)'])
            pressure_df = pd.DataFrame(pressure_data, index=['Pressure (ref_sim)'])
            head_df.to_csv(self.node_file, index=True, header=True)
            demand_df.to_csv(self.node_file, index=True, header=False)
            pressure_df.to_csv(self.node_file, index=True, header=False)

        elif data_type == 'resilience_simulation':
            head_index = list()
            demand_index = list()
            pressure_index = list()
            head_temp_list = list()
            demand_temp_list = list()
            pressure_temp_list = list()

            for case in data:
                head_data.clear()
                demand_data.clear()
                pressure_data.clear()
                head_index.append('Head (' + str(case['ID']) + ')')
                demand_index.append('Demand (' + str(case['ID']) + ')')
                pressure_index.append('Pressure (' + str(case['ID']) + ')')

                for node_id in self.wn.node_name_list:
                    head_data[node_id] = 0
                    demand_data[node_id] = 0
                    pressure_data[node_id] = 0

                for sim_step in case['head']:
                    for node_id in case['head'][sim_step]:
                        head_data[node_id] = head_data[node_id] + case['head'][sim_step][node_id]
                total_sum = 0
                for element in head_data.values():

                    total_sum = total_sum + element
                head_data['sum'] = total_sum
                head_data['percent'] = round((total_sum * 100) / self.sum_of_head, 2)

                for category in self.init_config.head_category:
                    if self.init_config.head_category[category][0] <= head_data['percent'] <=
                                                self.init_config.head_category[category][1]:
                        head_data['category'] = category
                        break
                head_temp_list.append(copy.deepcopy(head_data))
                head_data['remove_element_id'] = case['ID']
                head_data['element_type'] = case['type']
                self.nodes_results['head'].append(copy.deepcopy(head_data))

                for sim_step in case['demand']:
                    for node_id in case['demand'][sim_step]:
                        demand_data[node_id] = demand_data[node_id] +
                                                    case['demand'][sim_step][node_id]
                total_sum = 0
                for element in demand_data.values():
                    total_sum = total_sum + element
                demand_data['sum'] = total_sum
                demand_data['percent'] = round((total_sum * 100) / self.sum_of_demand, 2)

                for category in self.init_config.demand_category:
                    if self.init_config.demand_category[category][0] <= demand_data['percent'] <=
                                                self.init_config.demand_category[category][1]:
                        demand_data['category'] = category
                        break
                demand_temp_list.append(copy.deepcopy(demand_data))
                demand_data['remove_element_id'] = case['ID']
                demand_data['element_type'] = case['type']
                self.nodes_results['demand'].append(copy.deepcopy(demand_data))
```

```
                for sim_step in case['pressure']:
                    for node_id in case['pressure'][sim_step]:
                        pressure_data[node_id] = pressure_data[node_id] +
                                                 case['pressure'][sim_step][node_id]

                total_sum = 0
                for element in pressure_data.values():
                    total_sum = total_sum + element
                pressure_data['sum'] = total_sum
                pressure_data['percent'] = round((total_sum * 100) / self.sum_of_pressure, 2)

                for category in self.init_config.pressure_category:
                    if self.init_config.pressure_category[category][0] <= pressure_data['percent']
                                        <= self.init_config.pressure_category[category][1]:
                        pressure_data['category'] = category
                        break
                pressure_temp_list.append(copy.deepcopy(pressure_data))
                pressure_data['remove_element_id'] = case['ID']
                pressure_data['element_type'] = case['type']
                self.nodes_results['pressure'].append(copy.deepcopy(pressure_data))

            head_df = pd.DataFrame(head_temp_list, index=head_index)
            demand_df = pd.DataFrame(demand_temp_list, index=demand_index)
            pressure_df = pd.DataFrame(pressure_temp_list, index=pressure_index)
            head_df.to_csv(self.node_file, index=True, header=False)
            demand_df.to_csv(self.node_file, index=True, header=False)
            pressure_df.to_csv(self.node_file, index=True, header=False)

        return self.nodes_results

    def network_resilience_algorithm(self):
        ref_sim = self.reference_simulation()
        res_sim = self.reservoir_resilience_simulation()
        tan_sim = self.tank_resilience_simulation()
        pum_sim = self.pump_resilience_simulation()
        val_sim = self.valve_resilience_simulation()
        pip_sim = self.pipe_resilience_simulation()

        self.create_data_frame_for_links('reference_simulation', ref_sim)
        self.create_data_frame_for_nodes('reference_simulation', ref_sim)
        self.create_data_frame_for_links('resilience_simulation', res_sim)
        self.create_data_frame_for_nodes('resilience_simulation', res_sim)
        self.create_data_frame_for_links('resilience_simulation', tan_sim)
        self.create_data_frame_for_nodes('resilience_simulation', tan_sim)
        self.create_data_frame_for_links('resilience_simulation', pum_sim)
        self.create_data_frame_for_nodes('resilience_simulation', pum_sim)
        self.create_data_frame_for_links('resilience_simulation', val_sim)
        self.create_data_frame_for_nodes('resilience_simulation', val_sim)
        self.create_data_frame_for_links('resilience_simulation', pip_sim)
        self.create_data_frame_for_nodes('resilience_simulation', pip_sim)

        return {'links': self.links_results, 'nodes': self.nodes_results}


# ============================================================================== Main
# new_case = Algorithm_1()
# new_case.network_resilience_algorithm()
```

Załącznik D – Kod źródłowy Algorytmu 2

```
"""
          Name:     Algorithm 2- Algorytm wyznaczania tras przepływu wody
        Author:     Ariel Antonowicz
   Last update:     26.02.2022
"""

import wntr
import copy
import os, glob
from config import Configuration_data


class Algorithm_2:

    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.nodes_to_check = list()
        self.results = list()
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def run_simulation(self, wn, source):
        simulation_results = list()
        sim = wntr.sim.EpanetSimulator(wn=wn)

        wn.options.quality.parameter = 'TRACE'
        wn.options.quality.trace_node = source

        for step in range(0, self.init_config.sim_duration + 1, 1):
            wn.options.time.duration = step * 3600
            simulation_results = sim.run_sim()

        return {'quality': simulation_results.node['quality'].to_dict('index'),
                'flow': simulation_results.link['flowrate'].to_dict('index')}

    def check_neighbours(self, node_id):
        node_neighbours_list = list()

        for link in self.wn.link_name_list:
            if node_id == self.wn.get_link(link).todict()['start_node_name']:
                node_neighbours_list.append({'node_id': node_id, 'link_id': link,
                                             'start_node_id': node_id,
                                             'end_node_id': self.wn.get_link(link)
                                              .todict()['end_node_name']})

            elif node_id == self.wn.get_link(link).todict()['end_node_name']:
                node_neighbours_list.append({'node_id': node_id, 'link_id': link,
                                             'start_node_id': self.wn.get_link(link)
                                             .todict()['start_node_name'], 'end_node_id': node_id})

        return node_neighbours_list

    def water_trace_data(self, node_id):
        # Create source list:
        source_list = list(self.wn.nodes.reservoir_names)
        source_list.extend(list(self.wn.nodes.tank_names))

        result = list()

        self.nodes_to_check.append(node_id)

        for node in self.nodes_to_check:
            node_neighbours_list = self.check_neighbours(node)

            wn = wntr.network.WaterNetworkModel(self.inp_file)
            sim_result = self.run_simulation(wn, source=node)

            quality_results = copy.deepcopy(sim_result['quality'])
            flow_results = copy.deepcopy(sim_result['flow'])
```

```
            for step in quality_results:

                for neighbour in node_neighbours_list:
                    link_id = neighbour['link_id']
                    neighbour['sim_step'] = step
                    neighbour['flow'] = copy.deepcopy(flow_results[step][link_id])

                    if (neighbour['start_node_id'] == neighbour['node_id'])
                                                        and (neighbour['flow'] > 0):
                        neighbour['role'] = {'node_id': neighbour['node_id'],
                               'neighbour_id': neighbour['end_node_id'], 'status': 'giver'}
                    elif (neighbour['start_node_id'] == neighbour['node_id'])
                                                            and (neighbour['flow'] < 0):
                        neighbour['role'] = {'node_id': neighbour['node_id'],
                               'neighbour_id': neighbour['end_node_id'], 'status': 'receiver'}
                        if (neighbour['end_node_id'] not in source_list) and
                                    (neighbour['end_node_id'] not in self.nodes_to_check)
                                                        and neighbour['end_node_id'] != node:
                            self.nodes_to_check.append(neighbour['end_node_id'])
                    elif (neighbour['end_node_id'] == neighbour['node_id'])
                                                            and (neighbour['flow'] > 0):
                        neighbour['role'] = {'node_id': neighbour['node_id'],
                                'neighbour_id': neighbour['start_node_id'], 'status': 'receiver'}
                        if (neighbour['start_node_id'] not in source_list) and
                                (neighbour['start_node_id'] not in self.nodes_to_check)
                                                        and neighbour['start_node_id'] != node:
                            self.nodes_to_check.append(neighbour['start_node_id'])
                    else:
                        neighbour['role'] = {'node_id': neighbour['node_id'],
                          'neighbour_id': neighbour['start_node_id'], 'status': 'giver'}
                    result.append(copy.deepcopy(neighbour))
        return result

    def path_creator(self, node_id):
        data = self.water_trace_data(node_id)
        simple_info = list()
        node_list = list()
        givers = dict()
        path = dict()
        for case in data:
            if case['role']['status'] == 'receiver':
                if case['start_node_id'] not in node_list:
                    node_list.append(case['start_node_id'])
                if case['end_node_id'] not in node_list:
                    node_list.append(case['end_node_id'])

                simple_info.append({'step': case['sim_step'],
                                    'rec': case['node_id'],
                                    'giv': case['role']['neighbour_id'],
                                    'flow': case['flow'],
                                    'link_id': case['link_id']})
        for step in range(0, self.init_config.sim_duration * 3600, 3600):
            path[step] = list()
            for node in node_list:
                givers[node] = list()
                for item in simple_info:
                    if item['step'] == step:
                        if node == item['giv']:
                            givers[node].append([item['rec'], item['link_id']])
            path[step].append(givers)
        return path

    def critical_pipes(self, node_id):
        data = self.water_trace_data(node_id)
        pipe_id_list = list()
        result = dict()

        for case in data:
            if case['role']['status'] == 'receiver':
                if case['link_id'] not in pipe_id_list:
                    pipe_id_list.append(case['link_id'])
        for link in pipe_id_list:
            result[link] = 0

        for case in data:
            if case['link_id'] in pipe_id_list:
                result[case['link_id']] = result[case['link_id']] + abs(case['flow'])

        return result

# ============================================================================== Main
# new_case = Algorithm_2()
# new_case.path_creator(node_id='J14')
# new_case.critical_pipes('J14')
```

Załącznik E – Informacje o zasuwach w przykładowym modelu testowym


```
{
  "set_1":
     [
     {"valve_id": "V1", "data": {"node_id": "J1", "link_id": "P2", "coordinates": [33.0, 95.0]}},
     {"valve_id": "V2", "data": {"node_id": "J2", "link_id": "P3", "coordinates": [58.0, 95.0]}},
     {"valve_id": "V3", "data": {"node_id": "J2", "link_id": "P4", "coordinates": [62.0, 95.0]}},
     {"valve_id": "V4", "data": {"node_id": "J3", "link_id": "P5", "coordinates": [87.0, 95.0]}},
     {"valve_id": "V5", "data": {"node_id": "J3", "link_id": "P8", "coordinates": [85.0, 93.0]}},
     {"valve_id": "V6", "data": {"node_id": "J6", "link_id": "P15", "coordinates": [10.0, 82.0]}},
     {"valve_id": "V7", "data": {"node_id": "J7", "link_id": "P10", "coordinates": [33.0, 80.0]}},
     {"valve_id": "V8", "data": {"node_id": "J7", "link_id": "P6", "coordinates": [35.0, 82.0]}},
     {"valve_id": "V9", "data": {"node_id": "J7", "link_id": "P16", "coordinates": [35.0, 78.0]}},
     {"valve_id": "V10", "data": {"node_id": "J7", "link_id": "P11", "coordinates": [37.0, 80.0]}},
     {"valve_id": "V11", "data": {"node_id": "J9", "link_id": "P12", "coordinates": [83.0, 80.0]}},
     {"valve_id": "V12", "data": {"node_id": "J9", "link_id": "P18", "coordinates": [80.0, 82.0]}},
     {"valve_id": "V13", "data": {"node_id": "J10", "link_id": "P9", "coordinates": [110.0, 82.0]}},
     {"valve_id": "V14", "data": {"node_id": "J11", "link_id": "P14", "coordinates": [128.0, 80.0]}},
     {"valve_id": "V15", "data": {"node_id": "J12", "link_id": "P15", "coordinates": [10.0, 62.0]}},
     {"valve_id": "V16", "data": {"node_id": "J13", "link_id": "P22", "coordinates": [37.0, 60.0]}},
     {"valve_id": "V17", "data": {"node_id": "J14", "link_id": "P22", "coordinates": [58.0, 60.0]}},
     {"valve_id": "V18", "data": {"node_id": "J14", "link_id": "P17", "coordinates": [60.0, 62.0]}},
     {"valve_id": "V19", "data": {"node_id": "J14", "link_id": "P23", "coordinates": [62.0, 60.0]}},
     {"valve_id": "V20", "data": {"node_id": "J15", "link_id": "P28", "coordinates": [85.0, 58.0]}},
     {"valve_id": "V21", "data": {"node_id": "J16", "link_id": "P24", "coordinates": [108.0, 60.0]}},
     {"valve_id": "V22", "data": {"node_id": "J16", "link_id": "P19", "coordinates": [110.0, 62.0]}},
     {"valve_id": "V23", "data": {"node_id": "J16", "link_id": "P29", "coordinates": [110.0, 58.0]}},
     {"valve_id": "V24", "data": {"node_id": "J17", "link_id": "P25", "coordinates": [128.0, 60.0]}},
     {"valve_id": "V25", "data": {"node_id": "J21", "link_id": "P34", "coordinates": [60.0, 38.0]}},
     {"valve_id": "V26", "data": {"node_id": "J22", "link_id": "P32", "coordinates": [83.0, 40.0]}},
     {"valve_id": "V27", "data": {"node_id": "J22", "link_id": "P37", "coordinates": [80.0, 38.0]}},
     {"valve_id": "V28", "data": {"node_id": "J23", "link_id": "P30", "coordinates": [110.0, 42.0]}},
     {"valve_id": "V29", "data": {"node_id": "J24", "link_id": "P35", "coordinates": [60.0, 28.0]}},
     {"valve_id": "V30", "data": {"node_id": "J26", "link_id": "P35", "coordinates": [83.0, 20.0]}},
     {"valve_id": "V31", "data": {"node_id": "J27", "link_id": "P38", "coordinates": [110.0, 22.0]}}
     ]
}
```

Załącznik F – Kod źródłowy Algorytmu 3

```
"""
          Name:     Algorithm 3 - Algorytm typowania zasuw do zamknięcia
        Author:     Ariel Antonowicz
   Last update:     28.02.2022
"""

import wntr
import wntr.network.controls as controls
import pandas as pd
import itertools
import json
import os, glob
import time
import copy
import matplotlib.pyplot as plt
from config import Configuration_data


class Algorithm_3:

    def __init__(self):

        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        self.valve_info = self.init_config.valve_info_path + self.init_config.valve_info + '.json'
        self.segments = self.init_config.segments_info_path + self.init_config.segments + '.json'
        self.matrix_a = self.init_config.matrix_path + self.init_config.matrix_a + '.csv'
        self.matrix_b = self.init_config.matrix_path + self.init_config.matrix_b + '.csv'
        self.matrix_c = self.init_config.matrix_path + self.init_config.matrix_c + '.csv'

        with open(self.valve_info, "r") as jsonFile:
            self.valves = json.load(jsonFile)

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def valve_to_pipe_references(self, pipe_id):

        segment = self.find_segment(pipe_id)
        valve_to_pipe_reference = dict()

        for valve in segment['segment_valves']:
            valve_to_pipe_reference[valve] = list()
            for data in self.valves[self.init_config.valve_set]:
                if valve == data['valve_id']:
                    if data['data']['link_id'] in segment['segment_links']:
                        start_node = self.wn.get_link(data['data']['link_id'])
                                                                    .todict()['start_node']
                        end_node = self.wn.get_link(data['data']['link_id']).todict()['end_node']

                        if str(start_node) in segment['near_node_list']:
                            for link in self.wn.link_name_list:
                                if str(link) not in segment['segment_links']:
                                    if str(start_node) ==
                                       str(self.wn.get_link(link).todict()['start_node']) or
                                       str(start_node) == str(self.wn.get_link(link)
                                       .todict()['end_node']):
                                        valve_to_pipe_reference[valve].append(link)

                        if str(end_node) in segment['near_node_list']:
                            for link in self.wn.link_name_list:
                                if str(link) not in segment['segment_links']:
                                    if str(end_node) ==
                                    str(self.wn.get_link(link).todict()['start_node']) or
                                    str(end_node) == str(self.wn.get_link(link)
                                    .todict()['end_node']):
                                        valve_to_pipe_reference[valve].append(link)

                    else:
                        valve_to_pipe_reference[valve].append(data['data']['link_id'])
        return [valve_to_pipe_reference, segment['segment_valves']]
```

```
    def create_use_case(self, pipe_id):
        valve_to_pipe_ref, valves = self.valve_to_pipe_references(pipe_id)
        use_case_list = list()

        for valve in valves:
            temp = copy.deepcopy(valves)
            temp.remove(valve)
            use_case_list.append(temp)

        return [use_case_list, valve_to_pipe_ref]

    def min_number_of_valves(self, pipe_id):
        use_cases, ref = self.create_use_case(pipe_id)

        # Reference simulation:
        simulation_results = list()
        flow_result = list()
        pipe_to_close = list()
        ref_sum_open = 0
        ref_sum_close = 0
        use_cases_result= list()

        for pipe in ref.values():
            pipe_to_close.extend(pipe)

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        for step in range(0, self.init_config.sim_duration, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()
        flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

        for x in range(0, len(flow_result), 1):
            ref_sum_open = ref_sum_open + flow_result[x][x * 3600][pipe_id]

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)
        flow_result.clear()

        for pipe in pipe_to_close:
            act = controls.ControlAction(self.wn.get_link(pipe), 'status', 0)
            cond = controls.SimTimeCondition(self.wn, controls.Comparison.ge, '00:00:00')

            ctrl = controls.Control(cond, act)
            self.wn.add_control('control' + str(pipe), ctrl)

        for step in range(0, self.init_config.sim_duration, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()
        flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

        for x in range(0, len(flow_result), 1):
            ref_sum_close = ref_sum_close + flow_result[x][x * 3600][pipe_id]

        # Use cases sim:
        for case in use_cases:
            link_to_close = list()
            for valve in case:
                for val in ref:
                    if valve == val:
                        link_to_close.extend(ref[val])

            self.wn = wntr.network.WaterNetworkModel(self.inp_file)
            self.sim = wntr.sim.EpanetSimulator(wn=self.wn)
            flow_result.clear()

            for pipe in link_to_close:
                act = controls.ControlAction(self.wn.get_link(pipe), 'status', 0)
                cond = controls.SimTimeCondition(self.wn, controls.Comparison.ge, '00:00:00')

                ctrl = controls.Control(cond, act)
                self.wn.add_control('control' + str(pipe), ctrl)

            for step in range(0, self.init_config.sim_duration, 1):
                self.wn.options.time.duration = step * 3600
                simulation_results = self.sim.run_sim()
            flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

            flow = 0
            for x in range(0, len(flow_result), 1):
                flow = flow + flow_result[x][x * 3600][pipe_id]
            case.append(flow)
```

```
            if case[-1] <= 0:
                use_cases_result.append(case)

        if len(use_cases_result) == 0:
            return ref.keys()
        else:
            return use_cases_result[0][0:-1]

    def matrix_a_creator(self):
        matrix_a = pd.DataFrame([], self.wn.node_name_list, self.wn.link_name_list)

        for link in self.wn.link_name_list:
            for node in self.wn.node_name_list:
                if self.wn.links[self.wn.get_link(link)].todict()['start_node_name'] == str(node):
                    matrix_a.loc[node, link] = 1
                if self.wn.links[self.wn.get_link(link)].todict()['end_node_name'] == str(node):
                    matrix_a.loc[node, link] = 1
        matrix_a.to_csv(self.matrix_a)
        return matrix_a

    def matrix_b_creator(self):
        matrix_b = pd.DataFrame([], self.wn.node_name_list, self.wn.link_name_list)

        for link in self.wn.link_name_list:
            for node in self.wn.node_name_list:
                for valve in self.valves[self.init_config.valve_set]:
                    if (valve['data']['node_id'] == node) and (valve['data']['link_id'] == link):
                        matrix_b.loc[node, link] = 1

        matrix_b.to_csv(self.matrix_b)
        return matrix_b

    def matrix_c_creator(self):
        matrix_a = self.matrix_a_creator()
        matrix_b = self.matrix_b_creator()
        matrix_c = pd.DataFrame([], self.wn.node_name_list, self.wn.link_name_list)

        for link in self.wn.link_name_list:
            for node in self.wn.node_name_list:
                if matrix_a.loc[node, link] == 1 and matrix_b.loc[node, link] == 1:
                    pass
                elif matrix_a.loc[node, link] == 1 or matrix_b.loc[node, link] == 1:
                    matrix_c.loc[node, link] = 1

        matrix_c.to_csv(self.matrix_c)
        return matrix_c

    def search_in_row(self, node_id, matrix):
        results = list()
        for link in self.wn.link_name_list:
            if matrix.loc[node_id, link] == 1:
                results.append(link)
        return results

    def search_in_column(self, link_id, matrix):
        results = list()
        for node in self.wn.node_name_list:
            if matrix.loc[node, link_id] == 1:
                results.append(node)
        return results

    def find_segment(self, pipe_id):
        pipe_list = list()
        node_list = list()
        near_node_list = list()

        matrix_a = self.matrix_a_creator()
        matrix_b = self.matrix_b_creator()
        matrix_c = self.matrix_c_creator()

        pipe_list.append(pipe_id)

        # Find nodes and links ID belongs to segment
        for pipe in pipe_list:
            for res in self.search_in_column(pipe, matrix_c):
                if res not in node_list:
                    node_list.append(res)
            for node in node_list:
                for res in self.search_in_row(node, matrix_c):
                    if res not in pipe_list:
                        pipe_list.append(res)
```

```
        # Find segment near nodes
        for pipe in pipe_list:
            for res in self.search_in_column(pipe, matrix_a):
                if res not in node_list:
                    near_node_list.append(res)

        # Find valves needed to close analyzed segment
        valve_list = list()

        for node in node_list:
            for valve in self.valves[self.init_config.valve_set]:
                if valve['data']['node_id'] == node:
                    valve_list.append(valve['valve_id'])

        for node in near_node_list:
            for valve in self.valves[self.init_config.valve_set]:
                if (valve['data']['node_id'] == node) and (valve['data']['link_id'] in pipe_list):
                    valve_list.append(valve['valve_id'])

        return {'link_id': pipe_id, 'segment_links': pipe_list,
                'segment_nodes': node_list, 'near_node_list': near_node_list,
                'segment_valves': valve_list}

    def find_all_segments(self):
        link_list = self.wn.links.pipe_names
        segments_list = list()

        segment_id = 1

        for link in link_list:
            result = self.find_segment(link)

            count = 0
            for seg in segments_list:
                if result['link_id'] in seg['segment_links']:
                    count = count + 1
            del result['link_id']
            if count == 0:
                result['segment_id'] = 'S' + str(segment_id)
                segments_list.append(result)
                segment_id = segment_id + 1
        # segments_list.append({'valve_set_id': self.init_config.valve_set})
        return {'valve_set_id': self.init_config.valve_set, 'data': segments_list}

    def save_segments(self):
        data = self.find_all_segments()

        with open(self.segments, "w") as jsonFile:
            json.dump(data, jsonFile)


# ================================================================================ Main
# new_case = Algorithm_3()
# new_case.matrix_a_creator()
# new_case.matrix_b_creator()
# new_case.matrix_c_creator()
#
# new_case.find_all_segments()
# new_case.save_segments()
# new_case.min_number_of_valves('P38')
```

Załącznik G – Informacje o awariach w przykładowym modelu testowym


```
{"set_1":
     {"status": "NoUpdated", "data": [
                          {"failure_id": "fail_P11",
                           "failure_type": "F1",
                           "pipe_id": "P11",
                           "pipe_diameter":
                                 {"diameter_m": 0.609599, "diameter_in": 24, "type": "D1"},
                           "historical_data":
                                 {"number_of_failure": 10, "years": 10},
                           "failure_start": "00:00:00",
                           "type_of_substrate": ["paving_slabs", "asphalt"],
                           "place_of_occurrence": null,
                           "segment": {"link_id": "P11",
                                 "segment_links": ["P11", "P12", "P17", "P7"],
                                 "segment_nodes": ["J8", "J2"],
                                 "near_node_list": ["J7", "J9", "J14"],
                                 "segment_valves": ["V2", "V3", "V10", "V11", "V18"]},
                                 "element_criticality": [63.81, "CAT-3"],
                                  "damage_intensity_factor": 0.0033,
                                 "day_flow": 14.5004,
                                 "pipe_resilience_factor": 1.593,
                                 "failure_time_repair": ["17 godz. 27 min", 1047],
                                 "failure_class": "C1",
                                 "prioritization_factor": null},

                          {"failure_id": "fail_P38",
                            "failure_type": "F2",
                            "pipe_id": "P38",
                            "pipe_diameter":
                                 {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
                            "historical_data":
                                 {"number_of_failure": 3, "years": 8},
                            "failure_start": "00:00:00",
                            "type_of_substrate": ["paving_slabs", "asphalt"],
                            "place_of_occurrence": null,
                            "segment": {"link_id": "P38",
                            "segment_links": ["P38", "P33", "P28"],
                            "segment_nodes": ["J23", "J22"],
                            "near_node_list": ["J27", "J15"],
                            "segment_valves": ["V28", "V26", "V27", "V31", "V20"]},
                            "element_criticality": [97.26, "CAT-1"],
                            "damage_intensity_factor": 0.0012,
                            "day_flow": 0.8653,
                           "pipe_resilience_factor": 0.375,
                            "failure_time_repair": ["11 godz. 27 min", 687],
                            "failure_class": "C2", "prioritization_factor": null}]},
"set_2":
     {"status": "Updated", "data": [
                          {"failure_id": "fail_P2",
                          "failure_type": "F1",
                          "pipe_id": "P2",
                          "pipe_diameter":
                               {"diameter_m": 0.30479999, "diameter_in": 12, "type": "D3"},
                          "historical_data":
                               {"number_of_failure": 3, "years": 8},
                          "failure_start": "00:00:00",
                          "type_of_substrate": ["paving_slabs", "asphalt"],
                          "place_of_occurrence": "public",
                          "segment": {"link_id": "P2",
                          "segment_links": ["P2", "P10", "P40", "PUMP_1", "PUMP_2", "P1"],
                          "segment_nodes": ["J6", "J5", "R1"],
                          "near_node_list": ["J1", "J7"],
                          "segment_valves": ["V6", "V1", "V7"]},
                          "element_criticality": [106.05, "CAT-1"],
                          "damage_intensity_factor": 0.001,
                          "day_flow": -2.9055,
                          "pipe_resilience_factor": 0.136,
                          "failure_time_repair": ["7 godz. 34 min", 454],
                          "failure_class": "C3",
                          "prioritization_factor": 0},

                          {"failure_id": "fail_P3",
                          "failure_type": "F2",
                          "pipe_id": "P3",
                          "pipe_diameter":
                               {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
                          "historical_data":
                               {"number_of_failure": 3, "years": 8},
```

```
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P3",
"segment_links": ["P3", "P6"],
"segment_nodes": ["J1"],
"near_node_list": ["J2", "J7"],
"segment_valves": ["V1", "V2", "V8"]},
"element_criticality": [106.94, "CAT-1"],
"damage_intensity_factor": 0.0012,
"day_flow": -2.2953,
"pipe_resilience_factor": 0.104,
"failure_time_repair": ["10 godz. 16 min", 616],
"failure_class": "C2",
"prioritization_factor": 0},

{"failure_id": "fail_P5",
"failure_type": "F2",
"pipe_id": "P5",
"pipe_diameter":
      {"diameter_m": 0.3047, "diameter_in": 12, "type": "D3"},
"historical_data":
     {"number_of_failure": 9, "years": 5},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P5",
"segment_links": ["P5", "P9"],
"segment_nodes": ["J4"],
"near_node_list": ["J3", "J10"],
"segment_valves": ["V4", "V13"]},
"element_criticality": [101.17, "CAT-1"],
"damage_intensity_factor": 0.0059,
"day_flow": -0.7104,
"pipe_resilience_factor": 0.273,
"failure_time_repair": ["9 godz. 20 min", 560],
"failure_class": "C2",
"prioritization_factor": null},

{"failure_id": "fail_P9",
"failure_type": "F3",
"pipe_id": "P9",
"pipe_diameter":
     {"diameter_m": 0.254, "diameter_in": 10, "type": "D3"},
"historical_data":
     {"number_of_failure": 9, "years": 8},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P9",
"segment_links": ["P9", "P5"],
"segment_nodes": ["J4"],
"near_node_list": ["J10", "J3"],
"segment_valves": ["V13", "V4"]},
"element_criticality": [101.17, "CAT-1"],
"damage_intensity_factor": 0.0057,
"day_flow": 0.7104,
"pipe_resilience_factor": 0.222,
"failure_time_repair": ["10 godz. 20 min", 620],
"failure_class": "C2",
"prioritization_factor": null},

{"failure_id": "fail_P11",
"failure_type": "F2",
"pipe_id": "P11",
"pipe_diameter":
     {"diameter_m": 0.6095, "diameter_in": 24, "type": "D1"},
"historical_data":
     {"number_of_failure": 10, "years": 8},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P11",
"segment_links": ["P11", "P12", "P17", "P7"],
"segment_nodes": ["J8", "J2"],
"near_node_list": ["J7", "J9", "J14"],
"segment_valves": ["V2", "V3", "V10", "V11", "V18"]},
"element_criticality": [63.81, "CAT-3"],
"damage_intensity_factor": 0.0041,
"day_flow": 14.5004,
"pipe_resilience_factor": 1.593,
"failure_time_repair": ["24 godz. 15 min", 1455],
"failure_class": "C1",
"prioritization_factor": 1},
```

```
{"failure_id": "fail_P14",
"failure_type": "F2",
"pipe_id": "P14",
"pipe_diameter":
     {"diameter_m": 0.3047999, "diameter_in": 12, "type": "D3"},
"historical_data":
     {"number_of_failure": 9, "years": 5},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P14",
"segment_links": ["P14", "P13", "P19", "P8"],
"segment_nodes": ["J10", "J9"],
"near_node_list": ["J11", "J16", "J3"],
"segment_valves": ["V13", "V11", "V12", "V14", "V22", "V5"]},
"element_criticality": [97.81, "CAT-1"],
"damage_intensity_factor": 0.0059,
"day_flow": 0.5647,
"pipe_resilience_factor": 0.362,
"failure_time_repair": ["10 godz. 30 min", 630],
"failure_class": "C2",
"prioritization_factor": null},

{"failure_id": "fail_P18",
"failure_type": "F1",
"pipe_id": "P18",
"pipe_diameter":
     {"diameter_m": 0.60959, "diameter_in": 24, "type": "D1"},
"historical_data":
     {"number_of_failure": 7, "years": 4},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P18",
"segment_links": ["P18", "P24", "P23"],
"segment_nodes": ["J15"],
"near_node_list": ["J9", "J16", "J14"],
"segment_valves": ["V20", "V12", "V21", "V19"]},
"element_criticality": [89.7, "CAT-2"],
"damage_intensity_factor": 0.0057,
"day_flow": 6.8039,
"pipe_resilience_factor": 0.883,
"failure_time_repair": ["17 godz. 2 min", 1022],
"failure_class": "C1",
"prioritization_factor": 0},

{"failure_id": "fail_P20",
"failure_type": "F3",
"pipe_id": "P20",
"pipe_diameter":
     {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
"historical_data": {"number_of_failure": 6, "years": 5},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P20",
"segment_links": ["P20"],
"segment_nodes": ["J11", "J17"],
"near_node_list": [],
"segment_valves": ["V14", "V24"]},
"element_criticality": [100.08, "CAT-1"],
"damage_intensity_factor": 0.0039,
"day_flow": -0.0273,
"pipe_resilience_factor": 0.304,
"failure_time_repair": ["10 godz. 27 min", 627],
"failure_class": "C2",
"prioritization_factor": null},

{"failure_id": "fail_P21",
"failure_type": "F2",
"pipe_id": "P21",
"pipe_diameter":
     {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
"historical_data":
     {"number_of_failure": 5, "years": 8},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P21",
"segment_links": ["P21", "P16", "P26"],
"segment_nodes": ["J13", "J12", "J20"],
"near_node_list": ["J7"],
"segment_valves": ["V16", "V15", "V9"]},
```

```
"element_criticality": [101.62, "CAT-1"],
"damage_intensity_factor": 0.0021,
"day_flow": 2.5776,
"pipe_resilience_factor": 0.253,
"failure_time_repair": ["10 godz. 31 min", 631],
"failure_class": "C2", "prioritization_factor": 0},

{"failure_id": "fail_P23",
"failure_type": "F1",
"pipe_id": "P23",
"pipe_diameter":
     {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
"historical_data":
     {"number_of_failure": 6, "years": 10},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P23",
"segment_links": ["P23", "P24", "P18"],
"segment_nodes": ["J15"],
"near_node_list": ["J14", "J16", "J9"],
"segment_valves": ["V20", "V19", "V21", "V12"]},
"element_criticality": [99.14, "CAT-1"],
"damage_intensity_factor": 0.002,
"day_flow": 0.9655,
"pipe_resilience_factor": 0.324,
"failure_time_repair": ["7 godz. 25 min", 445],
"failure_class": "C3",
"prioritization_factor": 0},

{"failure_id": "fail_P33",
"failure_type": "F2",
"pipe_id": "P33",
"pipe_diameter":
     {"diameter_m": 0.4572, "diameter_in": 18, "type": "D2"},
"historical_data":
     {"number_of_failure": 5, "years": 9},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P33",
"segment_links": ["P33", "P28", "P38"],
"segment_nodes": ["J22", "J23"],
"near_node_list": ["J15", "J27"],
"segment_valves": ["V26", "V27", "V28", "V20", "V31"]},
"element_criticality": [99.21, "CAT-1"],
"damage_intensity_factor": 0.0018,
"day_flow": 0.5465,
"pipe_resilience_factor": 0.475,
"failure_time_repair": ["15 godz. 45 min", 945],
"failure_class": "C1",
"prioritization_factor": null},

{"failure_id": "fail_P37",
"failure_type": "F1",
"pipe_id": "P37",
"pipe_diameter":
     {"diameter_m": 0.4572, "diameter_in": 18, "type": "D2"},
"historical_data":
     {"number_of_failure": 5, "years": 4},
"failure_start": "00:00:00",
"type_of_substrate": ["paving_slabs", "asphalt"],
"place_of_occurrence": "public",
"segment": {"link_id": "P37", "
segment_links": ["P37", "P39"],
"segment_nodes": ["J26", "J27"],
"near_node_list": ["J22"],
"segment_valves": ["V30", "V31", "V27"]},
"element_criticality": [105.7, "CAT-1"],
"damage_intensity_factor": 0.0041,
"day_flow": -1.5484,
"pipe_resilience_factor": 0.29,
"failure_time_repair": ["11 godz. 52 min", 712],
"failure_class": "C2",
"prioritization_factor": 0},

{"failure_id": "fail_P39",
"failure_type": "F2",
"pipe_id": "P39",
"pipe_diameter":
     {"diameter_m": 0.30479, "diameter_in": 12, "type": "D3"},
"historical_data":
     {"number_of_failure": 7, "years": 5},
```

```
                    "failure_start": "00:00:00",
                    "type_of_substrate": ["paving_slabs", "asphalt"],
                    "place_of_occurrence": "public",
                    "segment": {"link_id": "P39",
                    "segment_links": ["P39", "P37"],
                    "segment_nodes": ["J26", "J27"],
                    "near_node_list": ["J22"],
                    "segment_valves": ["V30", "V31", "V27"]},
                    "element_criticality": [102.16, "CAT-1"],
                    "damage_intensity_factor": 0.0046,
                    "day_flow": -0.6922,
                    "pipe_resilience_factor": 0.241,
                    "failure_time_repair": ["10 godz. 24 min", 624],
                    "failure_class": "C2",
                    "prioritization_factor": null}]
    }
}
```

Załącznik H – Informacje dodatkowe o analizowanej sieci

```
{
    "set_1":
    {
        "critical_infrastructure_nodes": ["J2", "J15"],
        "delivery_time": 60,
        "suspended_time": 60,
        "failure_material_changer_probability": 0.15,
        "warehouse": [
            {"id": "J6",
            "data":{
                "max_capacity": {
                "clamps": 3000,
                "pipes": 3000,
                "other_stuff": 14000
                },
                "stock_levels": {
                    "clamps": {"D1": 1000, "D2": 1000, "D3": 1000},
                    "pipes": {"D1": 1000, "D2": 1000, "D3": 1000},
                    "other_stuff": 14000
                }}},
            {"id": "J23",
            "data":{
                "max_capacity": {
                "clamps": 3000,
                "pipes": 3000,
                "other_stuff": 14000
                },
                "stock_levels": {
                    "clamps": {"D1": 1000, "D2": 1000, "D3": 1000},
                    "pipes": {"D1": 1000, "D2": 1000, "D3": 1000},
                    "other_stuff": 14000
                }}}
    ]
}}
```

Załącznik I – Kod źródłowy Algorytmu 4

```
"""
          Name:     Algorithm 4 - Algorytm klasyfikacji awarii
        Author:     Ariel Antonowicz
   Last update:     23.03.2022
"""

import numpy as np
import random
from algorithms.algorithm_1.algorithm_1 import *
from algorithms.algorithm_3.algorithm_3 import *


class Algorithm_4:
    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        self.classification_results = list()

        self.failures_times = self.init_config.repair_times
        self.failures_types = self.init_config.failure_type
        self.failures_info = self.init_config.failures_path + self.init_config.failures_info +
                                                                                        '.json'
        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def update_data(self):

        if self.failures[self.init_config.failures_set]['status'] == 'NoUpdated':
            # Element critical factor:
            links_result = Algorithm_1().network_resilience_algorithm()

            for fail in self.failures[self.init_config.failures_set]['data']:
                # Diameter:
                diameter = self.wn.get_link(fail['pipe_id']).todict()['diameter']
                if diameter <= self.init_config.map_pipe_diameter['D3'][1]:
                    diam_type = 'D3'
                elif self.init_config.map_pipe_diameter['D2'][0] < diameter <=
                                                self.init_config.map_pipe_diameter['D2'][1]:
                    diam_type = 'D2'
                else:
                    diam_type = 'D1'
                fail['pipe_diameter'] = {'diameter_m': diameter,'diameter_in':
                                                round(diameter * 39.37), 'type': diam_type}

                # Type of substrate:
                if not fail['type_of_substrate']:
                    fail['type_of_substrate'] = ['paving_slabs', 'asphalt']

                # Historical data:
                if not fail['historical_data']:
                    fail['historical_data'] = {'number_of_failure':
                          random.randint(self.init_config.historical_data_random[0],
                          self.init_config.historical_data_random[1]), 'years':
                          random.randint(self.init_config.historical_data_random[0],
                          self.init_config.historical_data_random[1])}
                # Place occurrence:
                if not fail['place_of_occurrence']:
                    fail['place_of_occurrence'] = 'public'

                # Segment:
                fail['segment'] = Algorithm_3().find_segment(fail['pipe_id'])

                # Element critical factor:
                for link in links_result['links']:
                    for case in links_result['links'][link]:
                        if case['remove_element_id'] == fail['pipe_id']:
                            fail['element_criticality'] = [case['percent'], case['category']]
                            break
            # Update info:
            self.failures[self.init_config.failures_set]['status'] = 'Updated'
```

```
            with open(self.failures_info, "w") as jsonFile:
                json.dump(self.failures, jsonFile)

    def damage_intensity_factor(self, fail):              # Wskaźnik intesywności uszkodzeń (lambda)
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)
        total_length = 0
        pipe_length = 0

        for pipe in self.wn.link_name_list:
            if self.wn.links[pipe].todict()['link_type'] == 'Pipe':
                total_length = total_length + self.wn.links[pipe].todict()['length']
            if self.wn.links[pipe].todict()['name'] == fail['pipe_id']:
                pipe_length = self.wn.links[pipe].todict()['length']

        fail['damage_intensity_factor'] = round(fail['historical_data']['number_of_failure'] /
                                        (pipe_length * fail['historical_data']['years']), 4)
        return fail

    def day_flow(self, fail, sim_time=24):
        sum_of_flow = 0
        pipe_flow = 0
        flow_result = list()

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        for step in range(0, sim_time + 1, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()
            flow_result.append(simulation_results.link['flowrate'].to_dict('index'))

        for step in range(0, len(flow_result), 1):

            for key in flow_result[step][step*3600]:
                flow_result[step][step * 3600][key] =
                                        round(flow_result[step][step*3600][key] * 15.85, 2)
                sum_of_flow = sum_of_flow + flow_result[step][step * 3600][key]

                if key == fail['pipe_id']:
                    pipe_flow = pipe_flow + flow_result[step][step * 3600][key]

        fail['day_flow'] = round((round(pipe_flow, 2) * 100) / round(sum_of_flow, 2), 4)
        return fail

    def pipe_resilience_factor(self, fail, sim_time=24):
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        benchmark_pressure = list()
        benchmark_flow = list()
        benchmark_demand = list()
        benchmark_nodes_without_demand = list()
        benchmark_node_pressure_bellow_5 = list()
        bench_sum_of_flow = 0

        sim_pressure = list()
        sim_flow = list()
        sim_demand = list()
        sim_nodes_without_demand = list()
        sim_node_pressure_bellow_5 = list()
        sim_sum_of_flow = 0

        demand_result = list()
        pressure_result = list()

        for step in range(0, sim_time + 1, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()

            benchmark_flow.append(simulation_results.link['flowrate'].to_dict('index'))
            benchmark_demand.append(simulation_results.node['demand'].to_dict('index'))
            benchmark_pressure.append(simulation_results.node['pressure'].to_dict('index'))

            dem = 0
            pre = 0

            for node in self.wn.node_name_list:
                # print(benchmark_demand[-1][step*3600][node])
                # print(benchmark_pressure[-1][step * 3600][node])
                if benchmark_demand[-1][step*3600][node] <= 0.0:
```

```
                    dem = dem + 1
                if benchmark_pressure[-1][step*3600][node] <= 5.0:
                    pre = pre + 1
            benchmark_nodes_without_demand.append(dem)
            benchmark_node_pressure_bellow_5.append(pre)

        for step in range(0, len(benchmark_flow), 1):

            for key in benchmark_flow[step][step*3600]:
                benchmark_flow[step][step * 3600][key] =
                                    round(benchmark_flow[step][step*3600][key] * 15.85, 2)
                bench_sum_of_flow = bench_sum_of_flow + benchmark_flow[step][step * 3600][key]

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        act = controls.ControlAction(self.wn.get_link(fail['pipe_id']), 'status', 0)
        cond = controls.SimTimeCondition(self.wn, controls.Comparison.ge, '00:00:00')
        ctr1 = controls.Control(cond, act)
        self.wn.add_control('control' + str(fail['pipe_id']), ctr1)

        for step in range(0, sim_time + 1, 1):
            self.wn.options.time.duration = step * 3600
            simulation_results = self.sim.run_sim()

            sim_flow.append(simulation_results.link['flowrate'].to_dict('index'))
            sim_demand.append(simulation_results.node['demand'].to_dict('index'))
            sim_pressure.append(simulation_results.node['pressure'].to_dict('index'))

            dem = 0
            pre = 0

            for node in self.wn.node_name_list:
                if sim_demand[-1][step*3600][node] <= 0.0:
                    dem = dem + 1
                if sim_pressure[-1][step*3600][node] <= 5.0:
                    pre = pre + 1
            sim_nodes_without_demand.append(dem)
            sim_node_pressure_bellow_5.append(pre)

        for x in range(0, len(benchmark_nodes_without_demand), 1):
            demand_result.append(benchmark_nodes_without_demand[x] -
                                                    sim_nodes_without_demand[x])
            pressure_result.append(benchmark_node_pressure_bellow_5[x] -
                                                    sim_node_pressure_bellow_5[x])

        for step in range(0, len(sim_flow), 1):

            for key in sim_flow[step][step*3600]:
                sim_flow[step][step * 3600][key] = round(sim_flow[step][step*3600][key] * 15.85, 2)
                sim_sum_of_flow = sim_sum_of_flow + sim_flow[step][step * 3600][key]

        # print(f'Benchmark demand: {benchmark_nodes_without_demand}')
        # print(f'Sim demand      : {sim_nodes_without_demand}')
        # print(f'Result          : {demand_result}')
        # print(f'========================================')
        # print(f'Benchmark pressure: {benchmark_node_pressure_bellow_5}')
        # print(f'Sim pressure      : {sim_node_pressure_bellow_5}')
        # print(f'Result            : {pressure_result}')
        # print(f'========================================')
        # print(f'Benchmark flow    : {round(bench_sum_of_flow, 2)}')
        # print(f'Sim flow          : {round(sim_sum_of_flow, 2)}')
        # print(f'Result            : {round(bench_sum_of_flow - sim_sum_of_flow, 2)}')

        diameter = round(self.wn.links[self.wn.get_link(fail['pipe_id'])]
                                                .todict()['diameter'] * 39.73, 2)       # m to in
        flow_drop = round(bench_sum_of_flow - sim_sum_of_flow, 2)

        c_j = round((diameter + (sum(demand_result)/24 * -0.5) + (sum(pressure_result)/24 * -0.25)
                                                + flow_drop) / len(self.wn.pipe_name_list), 3)
        fail['pipe_resilience_factor'] = c_j
        return fail

    def todini_resilience_index(self):
        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.hydraulic.demand_model = 'PDD'
        sim = wntr.sim.WNTRSimulator(wn=self.wn)
        results = sim.run_sim()

        pressure = results.node['pressure']
        threshold = 21.09  # 30 psi
        pressure_above_threshold = wntr.metrics.query(pressure, np.greater, threshold)
        expected_demand = wntr.metrics.expected_demand(wn)
```

```
        demand = results.node['demand']
        wsa = wntr.metrics.water_service_availability(expected_demand, demand)
        head = results.node['head']
        pump_flowrate = results.link['flowrate'].loc[:, wn.pump_name_list]
        todini = wntr.metrics.todini_index(head, pressure, demand, pump_flowrate, wn, threshold)
        return todini

    def failure_time_repair(self, fail):
        t_factor = 0
        if fail['place_of_occurrence'] == 'private':
            t_factor = t_factor + 60
        for element in self.failures_types[fail['failure_type']]:
            needed_time = 0

            if element == 'open_valve' or element == 'close_valve':
                needed_time = len(fail['segment']['segment_valves']) *
                      random.randint(self.failures_times[fail['pipe_diameter']['type']][element][0],
                   self.failures_times[fail['pipe_diameter']['type']][element][1])
                t_factor = t_factor + needed_time

            elif (element == 'asphalt_extraction') and ('asphalt' not in
                                                        fail['type_of_substrate']):
                pass
            elif (element == 'paving_slabs') and ('paving_slabs' not in fail['type_of_substrate']):
                pass
            else:
                needed_time = random.randint(self.failures_times[fail['pipe_diameter']
                                                                 ['type']][element][0],

               self.failures_times[fail['pipe_diameter']['type']][element][1])
                t_factor = t_factor + needed_time
        hours = t_factor // 60
        minutes = t_factor % 60
        time_string = "{} godz. {} min".format(int(hours), int(minutes))

        fail['failure_time_repair'] = [time_string, t_factor]
        return fail

    def get_classify(self):
        # self.todini_resilience_index()
        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

        for fail in self.failures[self.init_config.failures_set]['data']:
            self.failure_time_repair(fail)
            self.damage_intensity_factor(fail)
            self.day_flow(fail)
            self.pipe_resilience_factor(fail)
            class_c1_sum = 0
            class_c2_sum = 0

            if fail['element_criticality'][1] == 'CAT-3' or
                                                 fail['element_criticality'][1] == 'CAT-4':
                class_c1_sum = class_c1_sum + 1
            if fail['element_criticality'][0] >= 15.0:
                class_c1_sum = class_c1_sum + 1
            if fail['pipe_resilience_factor'] >= 1.0:
                class_c1_sum = class_c1_sum + 1
            if fail['failure_time_repair'][1] >= 15*60:
                class_c1_sum = class_c1_sum + 1
            if fail['element_criticality'][1] == 'CAT-3' or
              fail['element_criticality'][1] == 'CAT-4' or fail['element_criticality'][1] == 'CAT-2':
                class_c2_sum = class_c2_sum + 1
            if fail['element_criticality'][0] >= 10.0:
                class_c2_sum = class_c2_sum + 1
            if fail['pipe_resilience_factor'] >= 0.7:
                class_c2_sum = class_c2_sum + 1
            if fail['failure_time_repair'][1] >= 8.5*60:
                class_c2_sum = class_c2_sum + 1
            if class_c1_sum >= 2:
                fail['failure_class'] = 'C1'
            elif class_c2_sum >= 2:
                fail['failure_class'] = 'C2'
            else:
                fail['failure_class'] = 'C3'
        with open(self.failures_info, "w") as jsonFile:
            json.dump(self.failures, jsonFile)
        return True

# ================================================================================ Main
# new_case = Algorithm_4()
# new_case.update_data()
# new_case.get_classify()
```

Załącznik J – Kod źródłowy Algorytmu 5

```
"""
          Name:     Algorithm 5 - Algorytm priorytetyzacji
        Author:     Ariel Antonowicz
   Last update:     24.03.2022
"""

from algorithms.algorithm_2.algorithm_2 import *
from algorithms.algorithm_4.algorithm_4 import *


class Algorithm_5:
    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        self.failures_info = self.init_config.failures_path + self.init_config.failures_info +
                                                                                        '.json'
        self.additional_info = self.init_config.additional_data_path +
                                                            self.init_config.additional_info + '.json'
        with open(self.additional_info, "r") as jsonFile:
            self.additional_info = json.load(jsonFile)

        self.failures = None
        self.trace = Algorithm_2()
        self.classification = Algorithm_4()

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def update_data(self):
        self.classification.update_data()
        self.classification.get_classify()

        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

        for fail in self.failures[self.init_config.failures_set]['data']:
            if fail['failure_type'] == 'F1':
                fail['prioritization_factor'] = 0
        return copy.deepcopy(self.failures)

    def get_failures_id_to_check(self):
        failure_to_check = list()

        for link in self.additional_info[self.init_config.additional_set]
                                                        ['critical_infrastructure_nodes']:
            links = self.trace.critical_pipes(link)
            for pipe in links:
                for fail in self.failures[self.init_config.failures_set]['data']:
                    if pipe == fail['pipe_id'] and fail['failure_type'] != 'F1':
                        failure_to_check.append({'critical_node_id': link,
                          'failure_id':fail['failure_id'], 'pipe_id': fail['pipe_id']})
        return copy.deepcopy(failure_to_check)

    def prioritization(self):
        reference_demand = dict()
        resilience_demand = dict()
        resilience_result = list()
        failures = list()
        failure_use_cases = list()

        temp = self.get_failures_id_to_check()
        temp2 = list()
        for item in temp:
            if item['pipe_id'] not in failures:
                failures.append(item['pipe_id'])

        for n in range(1, len(failures) + 1):
            temp2.append(list(set(list(itertools.combinations(failures, n)))))

        for i in range(0, len(temp2)):
            for j in range(0, len(temp2[i])):
                temp2[i][j] = list(temp2[i][j])
            failure_use_cases.extend(temp2[i])
```

```
        # Reference simulation:
        for node in self.additional_info[self.init_config.additional_set]
                                                        ['critical_infrastructure_nodes']:
            reference_demand[node] = 0

            self.wn = wntr.network.WaterNetworkModel(self.inp_file)
            self.wn.options.time.duration = self.init_config.sim_duration
            self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

            for step in range(0, self.init_config.sim_duration + 1, 1):
                self.wn.options.time.duration = step * 3600
                simulation_results = self.sim.run_sim()
                reference_demand[node] = reference_demand[node] +
                            simulation_results.node['demand'].to_dict('index')[step*3600][node]

        # Resilience simulation
        for fail in failure_use_cases:

            for node in self.additional_info[self.init_config.additional_set]
                                                        ['critical_infrastructure_nodes']:
                resilience_demand[node] = {'failure_case': fail, 'total_demand': 0}

            self.wn = wntr.network.WaterNetworkModel(self.inp_file)
            self.wn.options.time.duration = self.init_config.sim_duration
            self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

            for item in fail:
                act = controls.ControlAction(self.wn.get_link(item), 'status', 0)
                cond = controls.SimTimeCondition(self.wn, controls.Comparison.ge, '00:00:00')
                ctr1 = controls.Control(cond, act)
                self.wn.add_control('control' + str(item), ctr1)

            for step in range(0, self.init_config.sim_duration + 1, 1):
                self.wn.options.time.duration = step * 3600
                simulation_results = self.sim.run_sim()
                for node in self.additional_info[self.init_config.additional_set]
                                                        ['critical_infrastructure_nodes']:
                    resilience_demand[node]['total_demand'] =
                                resilience_demand[node]['total_demand'] +
                          simulation_results.node['demand'].to_dict('index')[step * 3600][node]

            resilience_result.append(copy.deepcopy(resilience_demand))

        for fail in self.failures[self.init_config.failures_set]['data']:
            if fail['failure_class'] == 'C1' and fail['pipe_id'] in failures:
                fail['prioritization_factor'] = 1

        with open(self.failures_info, "w") as jsonFile:
            json.dump(self.failures, jsonFile)
        return True


# ============================================================================== Main
# new_case = Algorithm_5()
# new_case.update_data()
# new_case.get_failures_id_to_check()
# new_case.prioritization()
```

Załącznik K – Kod źródłowy Algorytmu 6

```
"""
          Name:     Algorithm 6 - Algorytm agregacji awarii
        Author:     Ariel Antonowicz
   Last update:     26.03.2022
"""

from algorithms.algorithm_5.algorithm_5 import *


class Algorithm_6:
    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        self.failures = None
        self.failures_info = self.init_config.failures_path + self.init_config.failures_info +
                                                                                    '.json'
        self.segment_info = self.init_config.segments_info_path + self.init_config.segments +
                                                                                    '.json'
        self.aggregation_result = self.init_config.results_path +
                                                self.init_config.aggregation_result + '.json'

        self.classification_and_prioritization = Algorithm_5()

        with open(self.segment_info, "r") as jsonFile:
            self.segments = json.load(jsonFile)

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def update_data(self):
        self.classification_and_prioritization.update_data()
        self.classification_and_prioritization.prioritization()

        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

        return self.failures

    def aggregation_first_degree(self):
        aggregation_first_degree = list()

        for segment in self.segments['data']:
            temp = list()
            for fail in self.failures[self.init_config.failures_set]['data']:
                if fail['failure_type'] in ['F2', 'F3'] and fail['pipe_id'] in
                                                                    segment['segment_links']:
                    temp.append(fail['failure_id'])
            if len(temp) >= 2:
                aggregation_first_degree.append({'segment_id': segment['segment_id'],
                                                 'failures': temp})

        return {'aggregation_first_degree': aggregation_first_degree}

    def aggregation_second_degree(self):
        aggregation_second_degree = list()

        for fail_1 in self.failures[self.init_config.failures_set]['data']:
            for fail_2 in self.failures[self.init_config.failures_set]['data']:
                seg = list()
                if fail_1['failure_id'] != fail_2['failure_id']:
                    if fail_1['failure_type'] in ['F2', 'F3'] and fail_2['failure_type'] in
                                                                        ['F2', 'F3']:
                        if fail_1['failure_class'] != 'C1' and fail_2['failure_class'] != 'C1':
                            if any(item in fail_1['segment']['segment_valves'] for item in
                                                    fail_2['segment']['segment_valves']):
                                seg.clear()
                                for segment in self.segments['data']:
                                    if sorted(fail_1['segment']['segment_valves']) ==
                                                    sorted(segment['segment_valves']):
                                        seg.append(segment['segment_id'])
                                    if sorted(fail_2['segment']['segment_valves']) ==
                                                    sorted(segment['segment_valves']):
                                        seg.append(segment['segment_id'])
```

```
                                    if len(seg) >= 2 and seg[0] != seg[1]:
                                        aggregation_second_degree.append({'segment_id': seg,
                                         'failures': [fail_1['failure_id'], fail_2['failure_id']]})
                                        break
        result = list()
        for el in aggregation_second_degree:
            result.append({x: sorted(el[x]) for x in el.keys()})

        aggregation_second_degree.clear()

        for el in result:
            if el not in aggregation_second_degree:
                aggregation_second_degree.append(el)

        return {'aggregation_second_degree': aggregation_second_degree}

    def save_proposition_of_aggregation(self):

        first = self.aggregation_first_degree()
        second = self.aggregation_second_degree()
        aggregation = [first, second]

        with open(self.aggregation_result, "w") as jsonFile:
            json.dump(aggregation, jsonFile)

        return aggregation


# ================================================================================ Main
# new_case = Algorithm_6()
# new_case.update_data()
# new_case.aggregation_first_degree()
# new_case.aggregation_second_degree()
# new_case.save_proposition_of_aggregation()
```

Załącznik L – Zawartość scenariusza testowego


```
{
    "set_1":
    {
        "network_type": "unreal",
        "teams": [{
            "id": "RT1",
            "localisation": "J2",
            "work_time": 0,
            "max_capacity": {
                "clamps": 10,
                "pipes": 10,
                "other_stuff": 60
            },
            "stock_levels": {
                "clamps": {
                    "D1": 5,
                    "D2": 4,
                    "D3": 4
                },
                "pipes": {
                    "D1": 1,
                    "D2": 0,
                    "D3": 1
                },
                "other_stuff": 30
            }
        },
            {"id": "RT2",
            "localisation": "J16",
            "work_time": 0,
            "max_capacity": {
                "clamps": 15,
                "pipes": 15,
                "other_stuff": 60
            },
            "stock_levels": {
                "clamps": {
                    "D1": 2,
                    "D2": 2,
                    "D3": 2
                },
                "pipes": {
                    "D1": 2,
                    "D2": 2,
                    "D3": 2
                },
                "other_stuff": 25
            }
        },
            {"id": "RT3",
            "localisation": "J20",
            "work_time": 0,
            "max_capacity": {
                "clamps": 15,
                "pipes": 15,
                "other_stuff": 35
            },
            "stock_levels": {
                "clamps": {
                    "D1": 5,
                    "D2": 5,
                    "D3": 5
                },
                "pipes": {
                    "D1": 0,
                    "D2": 0,
                    "D3": 0
                },
                "other_stuff": 30
            }}]}
```

Załącznik M – Kod źródłowy Algorytmu 7

```
"""
          Name:     Algorithm 7 - Szeregowanie zadań ekip naprawczych
        Author:     Ariel Antonowicz
   Last update:     29.03.2022
"""

from algorithms.algorithm_6.algorithm_6 import *
from config import Configuration_data
import networkx as nx
import time


class Algorithm_7:
    def __init__(self):
        self.init_config = Configuration_data()
        self.inp_file = self.init_config.networks_path + self.init_config.inp_file_name + '.inp'

        self.aggregation_info =self.init_config.results_path + self.init_config.aggregation_result
                                                                                    + '.json'
        self.failures_info = self.init_config.failures_path + self.init_config.failures_info +
                                                                                    '.json'
        self.segment_info = self.init_config.segments_info_path + self.init_config.segments +
                                                                                            '.json'
        self.valve_info = self.init_config.valve_info_path + self.init_config.valve_info + '.json'

        self.wn = wntr.network.WaterNetworkModel(self.inp_file)
        self.wn.options.time.duration = self.init_config.sim_duration
        self.sim = wntr.sim.EpanetSimulator(wn=self.wn)

        self.total_paths = dict()
        self.reserved_teams = list()
        self.in_work = list()
        self.materials = None
        self.failures = None
        self.additional_info = None
        self.aggregation_result = None
        self.classification_and_prioritization = Algorithm_5()
        self.aggregation = Algorithm_6()

        self.scenario_info = self.init_config.scenarios_path + self.init_config.scenario_file +
                                                                                            '.json'
        with open(self.scenario_info, "r") as jsonFile:
            self.scenario_info = json.load(jsonFile)

        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

        with open(self.valve_info, "r") as jsonFile:
            self.valves = json.load(jsonFile)

        with open(self.segment_info, "r") as jsonFile:
            self.segments = json.load(jsonFile)
        self.additional_info = self.init_config.additional_data_path +
                               self.init_config.additional_info + '.json'
        with open(self.additional_info, "r") as jsonFile:
            self.additional_info = json.load(jsonFile)
        self.scenario = self.create_scenario()
        self.task_scheduling = dict()
        self.total_work_time = dict()
        self.number_of_task = dict()
        self.done_failures = list()
        self.graph, self.weights = self.graph_with_weights()
        self.available_repair_teams = len(self.scenario['repair_teams'])

    def __del__(self):
        for filename in glob.glob("temp*"):
            os.remove(filename)

    def update_data(self):
        self.classification_and_prioritization.update_data()

        with open(self.failures_info, "r") as jsonFile:
            self.failures = json.load(jsonFile)

        self.aggregation.save_proposition_of_aggregation()
        self.additional_info = self.init_config.additional_data_path +
                    self.init_config.additional_info + '.json'
        with open(self.additional_info, "r") as jsonFile:
            self.additional_info = json.load(jsonFile)
```

```
        return self.failures

    def create_levels(self, aggregation=True):

        levels = {'L1': list(), 'L2': list(), 'L3': list(), 'L4': list()}
        remove_list = list()
        failures = copy.deepcopy(self.failures[self.init_config.failures_set]['data'])

        if aggregation:
            aggregated_pipes = list()

            with open(self.aggregation_info, "r") as jsonFile:
                self.aggregation_result = json.load(jsonFile)

            for fail in self.aggregation_result[0]['aggregation_first_degree']:

                if fail['failures'][0][5:] and fail['failures'][1][5:] not in aggregated_pipes:
                    aggregated_pipes.append(fail['failures'][0][5:])
                    aggregated_pipes.append(fail['failures'][1][5:])

                    new_fail = dict()
                    repair_time = list()
                    classification = list()

                    new_fail['failure_id'] = 'AG_' + fail['failures'][0] + fail['failures'][1][4:]
                    new_fail['pipe_id'] = [fail['failures'][0][5:], fail['failures'][1][5:]]
                    new_fail['failure_type'] = list()
                    new_fail['pipe_diameter'] = list()
                    new_fail['historical_data'] = list()
                    new_fail['failure_start'] = list()
                    new_fail['type_of_substrate'] = list()
                    new_fail['place_of_occurrence'] = list()
                    new_fail['segment'] = dict()
                    new_fail['element_criticality'] = list()
                    new_fail['damage_intensity_factor'] = list()
                    new_fail['day_flow'] = list()
                    new_fail['pipe_resilience_factor'] = list()
                    new_fail['failure_time_repair'] = None
                    new_fail['failure_class'] = None
                    new_fail['prioritization_factor'] = 0
                    duplicate_valves = list()
                    time_to_reduce = 0

                    for f in failures:
                        if f['pipe_id'] in new_fail['pipe_id']:
                            new_fail['failure_type'].append(f['failure_type'])
                            new_fail['pipe_diameter'].append(f['pipe_diameter'])
                            new_fail['historical_data'].append(f['historical_data'])
                            new_fail['failure_start'].append(f['failure_start'])
                            new_fail['type_of_substrate'].append(f['type_of_substrate'])
                            new_fail['place_of_occurrence'].append(f['place_of_occurrence'])
                            new_fail['segment'] = f['segment']
                            duplicate_valves = f['segment']['segment_valves']
                            new_fail['element_criticality'].append(f['element_criticality'])
                         new_fail['damage_intensity_factor'].append(f['damage_intensity_factor'])
                            new_fail['day_flow'].append(f['day_flow'])
                            new_fail['pipe_resilience_factor'].append(f['pipe_resilience_factor'])
                            repair_time.append(copy.deepcopy(f['failure_time_repair']))
                            classification.append(copy.deepcopy(f['failure_class']))

                            if f not in remove_list:
                                remove_list.append(f)

                    if 'C2' in classification:
                        new_fail['failure_class'] = 'C2'
                    else:
                        new_fail['failure_class'] = 'C3'

                    for valve in duplicate_valves:
                        for val in self.valves[self.init_config.valve_set]:
                            if valve == val['valve_id']:
                                # Diameter:
                                diameter = self.wn.get_link(val['data']['link_id'])
                                                                    .todict()['diameter']
                                if diameter <= self.init_config.map_pipe_diameter['D3'][1]:
                                    diam_type = 'D3'
                                elif self.init_config.map_pipe_diameter['D2'][0] < diameter <=
                                                        self.init_config.map_pipe_diameter['D2'][1]:
                                    diam_type = 'D2'
```

```
                                else:
                                    diam_type = 'D1'

                                time_to_reduce = time_to_reduce +
                            random.randint(self.init_config.repair_times[diam_type]['open_valve'][0],

                            self.init_config.repair_times[diam_type]['open_valve'][1])

                                time_to_reduce = time_to_reduce +
                            random.randint(self.init_config.repair_times[diam_type]['close_valve'][0],

                            self.init_config.repair_times[diam_type]['close_valve'][1])

                            total_repair_time = repair_time[0][1] + repair_time[1][1] - time_to_reduce
                            hours = total_repair_time // 60
                            minutes = total_repair_time % 60
                            time_string = "{} godz. {} min".format(int(hours), int(minutes))

                            new_fail['failure_time_repair'] = [time_string, total_repair_time]
                            failures.append(new_fail)

                    for fail in self.aggregation_result[1]['aggregation_second_degree']:
                        if fail['failures'][0][5:] and fail['failures'][1][5:] not in aggregated_pipes:
                            aggregated_pipes.append(fail['failures'][0][5:])
                            aggregated_pipes.append(fail['failures'][1][5:])
                            new_fail = dict()
                            repair_time = list()
                            classification = list()
                            segment_info = list()
                            new_fail['failure_id'] = 'AG_' + fail['failures'][0] + fail['failures'][1][4:]
                            new_fail['pipe_id'] = [fail['failures'][0][5:], fail['failures'][1][5:]]
                            new_fail['failure_type'] = list()
                            new_fail['pipe_diameter'] = list()
                            new_fail['historical_data'] = list()
                            new_fail['failure_start'] = list()
                            new_fail['type_of_substrate'] = list()
                            new_fail['place_of_occurrence'] = list()
                            new_fail['segment'] = dict()
                            new_fail['segment']['link_id'] = list()
                            new_fail['segment']['segment_links'] = list()
                            new_fail['segment']['segment_nodes'] = list()
                            new_fail['segment']['near_node_list'] = list()
                            new_fail['segment']['segment_valves'] = list()
                            new_fail['element_criticality'] = list()
                            new_fail['damage_intensity_factor'] = list()
                            new_fail['day_flow'] = list()
                            new_fail['pipe_resilience_factor'] = list()
                            new_fail['failure_time_repair'] = None
                            new_fail['failure_class'] = None
                            new_fail['prioritization_factor'] = 0

                            for f in failures:
                                if f['pipe_id'] in new_fail['pipe_id']:
                                    new_fail['failure_type'].append(f['failure_type'])
                                    new_fail['pipe_diameter'].append(f['pipe_diameter'])
                                    new_fail['historical_data'].append(f['historical_data'])
                                    new_fail['failure_start'].append(f['failure_start'])
                                    new_fail['type_of_substrate'].append(f['type_of_substrate'])
                                    new_fail['place_of_occurrence'].append(f['place_of_occurrence'])
                                    new_fail['element_criticality'].append(f['element_criticality'])

                                 new_fail['damage_intensity_factor'].append(f['damage_intensity_factor'])
                                    new_fail['day_flow'].append(f['day_flow'])
                                    new_fail['pipe_resilience_factor'].append(f['pipe_resilience_factor'])
                                    repair_time.append(copy.deepcopy(f['failure_time_repair']))
                                    classification.append(copy.deepcopy(f['failure_class']))
                                    segment_info.append(copy.deepcopy(f['segment']))

                                    if f not in remove_list:
                                        remove_list.append(f)
                            duplicate_valves = list()
                            for seg in segment_info:
                                new_fail['segment']['link_id'].append(seg['link_id'])
                                for item in seg['segment_links']:
                                    if item not in new_fail['segment']['segment_links']:
                                        new_fail['segment']['segment_links'].append(item)
                                for item in seg['segment_nodes']:
                                    if item not in new_fail['segment']['segment_nodes']:
                                        new_fail['segment']['segment_nodes'].append(item)
                                for item in seg['near_node_list']:
                                    if item not in new_fail['segment']['near_node_list']:
                                        new_fail['segment']['near_node_list'].append(item)
                                for item in seg['segment_valves']:
```

```
                            if item not in new_fail['segment']['segment_valves']:
                                new_fail['segment']['segment_valves'].append(item)
                            else:
                                duplicate_valves.append(item)

                    if 'C2' in classification:
                        new_fail['failure_class'] = 'C2'
                    else:
                        new_fail['failure_class'] = 'C3'

                    time_to_reduce = 0

                    for valve in duplicate_valves:
                        for val in self.valves[self.init_config.valve_set]:
                            if valve == val['valve_id']:
                                # Diameter:
                                diameter =
                               self.wn.get_link(val['data']['link_id']).todict()['diameter']
                                if diameter <= self.init_config.map_pipe_diameter['D3'][1]:
                                    diam_type = 'D3'
                                elif self.init_config.map_pipe_diameter['D2'][0] < diameter <=
                                                    self.init_config.map_pipe_diameter['D2'][1]:
                                    diam_type = 'D2'
                                else:
                                    diam_type = 'D1'

                                time_to_reduce = time_to_reduce +
                            random.randint(self.init_config.repair_times[diam_type]['open_valve'][0],

                            self.init_config.repair_times[diam_type]['open_valve'][1])
                                time_to_reduce = time_to_reduce +
                        random.randint(self.init_config.repair_times[diam_type]['close_valve'][0],

                       self.init_config.repair_times[diam_type]['close_valve'][1])

                        total_repair_time = repair_time[0][1] + repair_time[1][1] - time_to_reduce
                        hours = total_repair_time // 60
                        minutes = total_repair_time % 60
                        time_string = "{} godz. {} min".format(int(hours), int(minutes))

                        new_fail['failure_time_repair'] = [time_string, total_repair_time]
                        failures.append(new_fail)

        for item in remove_list:
            failures.remove(item)

        for fail in failures:
            if fail['failure_id'][:3] == 'AG_':
                start_node_id = {'id': str(self.wn.get_node(self.wn.
                                    get_link(fail['pipe_id'][0]).todict()['start_node_name'])),
                                 'coordinates': [
                                    self.wn.get_node(self.wn.get_link(fail['pipe_id'][0])
                                    .todict()['start_node_name']).todict()['coordinates'][0],

            self.wn.get_node(self.wn.get_link(fail['pipe_id'][0])
                                    .todict()['start_node_name']).todict()['coordinates'][1]]}

                pipe_type = [fail['pipe_diameter'][0]['type'], fail['pipe_diameter'][0]['type']]
            else:
                start_node_id = {'id': str(self.wn.get_node(self.wn.
                                  get_link(fail['pipe_id']).todict()['start_node_name'])),
                                 'coordinates': [
                                  self.wn.get_node(self.wn.get_link(fail['pipe_id'])
                                   .todict()['start_node_name']).todict()['coordinates'][0],

                                    self.wn.get_node(self.wn.get_link(fail['pipe_id'])
                                    .todict()['start_node_name']).todict()['coordinates'][1]]}
                pipe_type = fail['pipe_diameter']['type']

            if fail['failure_type'] == 'F1':
                valves_id = None
            else:
                valves_id = dict()
                for segment in self.segments['data']:
                    if fail['pipe_id'] in segment['segment_links']:
                        temp = list()
                        for valve in segment['segment_valves']:

                            for data in self.valves[self.init_config.valve_set]:
                                if valve == data['valve_id']:
                                    temp.append({'valve_id': valve, 'node_id':
```

```
                    data['data']['node_id'], 'coordinates': data['data']['coordinates']})

                        valves_id['segment_id'] = {'segment_id': segment['segment_id'], 'valves': \
                                                   temp}

            if fail['prioritization_factor'] == 1:
                levels['L1'].append({'failure_id': fail['failure_id'],
                                     'failure_category': fail['failure_class'],
                                     'pipe_id': fail['pipe_id'],
                                     'pipe_type': pipe_type,
                                     'start_node': start_node_id,
                                     'failure_type': fail['failure_type'],
                                     'valves_id': valves_id,
                                     'failure_time_repair': fail['failure_time_repair']})

            elif fail['failure_class'] == 'C1':
                levels['L2'].append({'failure_id': fail['failure_id'],
                                     'failure_category': fail['failure_class'],
                                     'pipe_id': fail['pipe_id'],
                                     'pipe_type': pipe_type,
                                     'start_node': start_node_id,
                                     'failure_type': fail['failure_type'],
                                     'valves_id': valves_id,
                                     'failure_time_repair': fail['failure_time_repair']})

            elif fail['failure_class'] == 'C2':
                levels['L3'].append({'failure_id': fail['failure_id'],
                                     'failure_category': fail['failure_class'],
                                     'pipe_id': fail['pipe_id'],
                                     'pipe_type': pipe_type,
                                     'start_node': start_node_id,
                                     'failure_type': fail['failure_type'],
                                     'valves_id': valves_id,
                                     'failure_time_repair': fail['failure_time_repair']})
            else:
                levels['L4'].append({'failure_id': fail['failure_id'],
                                     'failure_category': fail['failure_class'],
                                     'pipe_id': fail['pipe_id'],
                                     'pipe_type': pipe_type,
                                     'start_node': start_node_id,
                                     'failure_type': fail['failure_type'],
                                     'valves_id': valves_id,
                                     'failure_time_repair': fail['failure_time_repair']})
        return levels

    def failure_repair_cost(self):
        levels = self.create_levels()
        total_cost = {'clamps': 0, 'pipes': 0, 'other_stuff': 0}

        for lev in levels:
            for fail in levels[lev]:
                if isinstance(fail['failure_type'], list):
                    clamps = 0
                    pipe = 0
                    other_stuff = 0
                    temp = list()
                    for kind in fail['failure_type']:
                        if kind == 'F1':
                            total_cost['clamps'] = total_cost['clamps'] +
                                                   self.init_config.repair_cost['F1']['clamps']
                            total_cost['pipes'] = total_cost['pipes'] +
                                                  self.init_config.repair_cost['F1']['pipes']
                            total_cost['other_stuff'] = total_cost['other_stuff'] +
                                                        self.init_config.repair_cost['F1']['other_stuff']
                            temp.append({'clamps': self.init_config.repair_cost['F1']['clamps'],
                            'pipes': self.init_config.repair_cost['F1']['pipes'], 'other_stuff':
                             self.init_config.repair_cost['F1']['other_stuff']})
                        elif kind == 'F2':
                            total_cost['clamps'] = total_cost['clamps'] +
                                                   self.init_config.repair_cost['F2']['clamps']
                            total_cost['pipes'] = total_cost['pipes'] +
                                                  self.init_config.repair_cost['F2']['pipes']
                            total_cost['other_stuff'] = total_cost['other_stuff'] +
                                                        self.init_config.repair_cost['F2']['other_stuff']
                            temp.append({'clamps': self.init_config.repair_cost['F2']['clamps'],
                            'pipes': self.init_config.repair_cost['F2']['pipes'], 'other_stuff':
                             self.init_config.repair_cost['F2']['other_stuff']})
                        else:
                            total_cost['clamps'] = total_cost['clamps'] +
                                                   self.init_config.repair_cost['F3']['clamps']
                            total_cost['pipes'] = total_cost['pipes'] +
```

```
                        self.init_config.repair_cost['F3']['pipes']
                        total_cost['other_stuff'] = total_cost['other_stuff'] +
                                                self.init_config.repair_cost['F3']['other_stuff']
                        temp.append({'clamps': self.init_config.repair_cost['F3']['clamps'],
                        'pipes': self.init_config.repair_cost['F3']['pipes'], 'other_stuff':
                         self.init_config.repair_cost['F3']['other_stuff']})
                    fail['repair_costs'] = temp

            else:
                if fail['failure_type'] == 'F1':
                    total_cost['clamps'] = total_cost['clamps'] +
                                                self.init_config.repair_cost['F1']['clamps']
                    total_cost['pipes'] = total_cost['pipes'] +
                                                 self.init_config.repair_cost['F1']['pipes']
                    total_cost['other_stuff'] = total_cost['other_stuff'] +
                                               self.init_config.repair_cost['F1']['other_stuff']
                    fail['repair_costs'] = {'clamps':
                                                 self.init_config.repair_cost['F1']['clamps'],
                         'pipes': self.init_config.repair_cost['F1']['pipes'], 'other_stuff':
                                              self.init_config.repair_cost['F1']['other_stuff']}
                elif fail['failure_type'] == 'F2':
                    total_cost['clamps'] = total_cost['clamps'] +
                                                self.init_config.repair_cost['F2']['clamps']
                    total_cost['pipes'] = total_cost['pipes'] +
                                                 self.init_config.repair_cost['F2']['pipes']
                    total_cost['other_stuff'] = total_cost['other_stuff'] +
                                               self.init_config.repair_cost['F2']['other_stuff']
                    fail['repair_costs'] = {'clamps':
                                                self.init_config.repair_cost['F2']['clamps'],
                        'pipes': self.init_config.repair_cost['F2']['pipes'], 'other_stuff':
                                               self.init_config.repair_cost['F2']['other_stuff']}
                else:
                    total_cost['clamps'] = total_cost['clamps'] +
                                                self.init_config.repair_cost['F3']['clamps']
                    total_cost['pipes'] = total_cost['pipes'] +
                                                self.init_config.repair_cost['F3']['pipes']
                    total_cost['other_stuff'] = total_cost['other_stuff'] +
                                               self.init_config.repair_cost['F3']['other_stuff']
                    fail['repair_costs'] = {'clamps':
                                                self.init_config.repair_cost['F3']['clamps'],
                         'pipes': self.init_config.repair_cost['F3']['pipes'], 'other_stuff':
                          self.init_config.repair_cost['F3']['other_stuff']}

        return levels, total_cost

    def create_scenario(self):
        levels, total_cost = self.failure_repair_cost()
        critical_infrastructure = self.additional_info[self.init_config.additional_set]
                                                          ['critical_infrastructure_nodes']
        delivery_time = self.additional_info[self.init_config.additional_set]['delivery_time']
        suspended_time = self.additional_info[self.init_config.additional_set]['suspended_time']
        failure_material_changer_probability =
                                          self.additional_info[self.init_config.additional_set]
                                                  ['failure_material_changer_probability']
        warehouses = self.additional_info[self.init_config.additional_set]['warehouse']
        repair_teams = self.scenario_info[self.init_config.scenario_set]['teams']
        temp_time = 0

        for le in levels:
            for fail in levels[le]:
                temp_time = temp_time + fail['failure_time_repair'][1]

        hours = temp_time // 60
        minutes = temp_time % 60
        time_string = "{} godz. {} min".format(int(hours), int(minutes))
        total_repair_time = [time_string, temp_time]

        return {'levels': levels,
                'total_repair_cost': total_cost,
                'critical_infrastructure': critical_infrastructure,
                'delivery_time': delivery_time,
                'suspended_time': suspended_time,
                'failure_material_changer_probability': failure_material_changer_probability,
                'warehouses': warehouses,
                'repair_teams': repair_teams,
                'total_repair_time': total_repair_time}

    def graph_with_weights(self, plot=False):
        length = self.wn.query_link_attribute('length')
        graph_with_weight = self.wn.get_graph(self.wn, link_weight=length)
        graph_with_weight = graph_with_weight.to_undirected()
        pos = nx.get_node_attributes(graph_with_weight, 'pos')
```

```
        weights = nx.get_edge_attributes(graph_with_weight, 'weight')
        new_weights = dict()

        for x in weights:
            new_weights[(x[0], x[1])] = weights[x]

        nx.draw_networkx_edge_labels(graph_with_weight, pos, edge_labels=new_weights)
        nx.draw_networkx_labels(graph_with_weight, pos, font_size=10, font_color='white')
        nx.draw(graph_with_weight, pos, node_size=400)
        if plot:
            plt.show()
        return graph_with_weight, new_weights

    @staticmethod
    def short_path_finder(graph, team, source):
        path = nx.bidirectional_dijkstra(graph, team, source, weight='weight')
        return path

    def needed_material(self):
        self.materials = {'clamps': {'D1': 0, 'D2': 0, 'D3':0}, 'pipes':
                                                {'D1': 0, 'D2': 0, 'D3':0}, 'other_stuff': 0 }

        for level in self.scenario['levels']:
            for fail in self.scenario['levels'][level]:
                if fail['failure_id'][0:3] == 'AG_':

                    self.materials['clamps'][fail['pipe_type'][0]] =
                                        self.materials['clamps'][fail['pipe_type'][0]]
                                        + fail['repair_costs'][0]['clamps']
                    self.materials['clamps'][fail['pipe_type'][1]] =
                                        self.materials['clamps'][fail['pipe_type'][1]] +
                                                        fail['repair_costs'][1]['clamps']
                    self.materials['pipes'][fail['pipe_type'][0]] =
                                        self.materials['pipes'][fail['pipe_type'][0]] +
                                                         fail['repair_costs'][0]['pipes']
                    self.materials['pipes'][fail['pipe_type'][1]] =
                                        self.materials['pipes'][fail['pipe_type'][1]] +
                                                         fail['repair_costs'][1]['pipes']
                    self.materials['other_stuff'] = self.materials['other_stuff'] +
                                        fail['repair_costs'][0]['other_stuff'] +
                                        fail['repair_costs'][1]['other_stuff']
                else:
                    self.materials['clamps'][fail['pipe_type']] =
                    self.materials['clamps'][fail['pipe_type']] + fail['repair_costs']['clamps']
                    self.materials['pipes'][fail['pipe_type']] =
                    self.materials['pipes'][fail['pipe_type']] + fail['repair_costs']['pipes']
                    self.materials['other_stuff'] = self.materials['other_stuff'] +
                                                        fail['repair_costs']['other_stuff']

        return self.materials

    def go_to_warehouse(self, team_id, failures_to_do, show_info=True):
        warehouse_path = list()

        for team in self.scenario['repair_teams']:
            self.number_of_task[team['id']] = len(self.task_scheduling[team['id']])
            self.total_work_time[team['id']] = team['work_time']

        for ware in self.scenario['warehouses']:
            for team2 in self.scenario['repair_teams']:
                if team2['id'] == team_id:
                    warehouse_path.append({'id': team_id, 'path':
                    self.short_path_finder(self.graph, team2['localisation'], ware['id'])})

        ware_minimum = {'id': None, 'path': (float('inf'), list())}
        for x in warehouse_path:
            if x['path'][0] < ware_minimum['path'][0]:
                ware_minimum = x

        self.task_scheduling[ware_minimum['id']].append(ware_minimum['path'][1][-1])
        self.total_paths[ware_minimum['id']].append(ware_minimum['path'][1])

        for team3 in self.scenario['repair_teams']:
            if ware_minimum['id'] == team3['id']:
                team3['localisation'] = ware_minimum['path'][1][-1]
                team3['work_time'] = round(team3['work_time'] + ware_minimum['path'][0] / 10, 0)

        path_form_warehouse_to_failure = list()
        for fail in failures_to_do:
            path_form_warehouse_to_failure.append({'failure_id': fail['failure_id'], 'path':
            self.short_path_finder(self.graph, ware_minimum['path'][1][-1],
            fail['start_node']['id'])})
```

```
        path_form_warehouse_to_failure = sorted(path_form_warehouse_to_failure,
                                                                key=lambda d: d['path'][0])

        for fail in failures_to_do:
            if fail['failure_id'] == path_form_warehouse_to_failure[0]['failure_id']:
                needed_stuff = {'clamps': {'D1': 0, 'D2':0, 'D3': 0},
                                'pipes': {'D1': 0, 'D2':0, 'D3': 0}, 'other_stuff': 0}
                if fail['failure_id'][0:3] == 'AG_':
                    for x in range(0, len(fail['pipe_type'])):
                        needed_stuff['clamps'][fail['pipe_type'][x]] =
                       needed_stuff['clamps'][fail['pipe_type'][x]] + fail['repair_costs'][x]['clamps']
                        needed_stuff['pipes'][fail['pipe_type'][x]] =
                       needed_stuff['pipes'][fail['pipe_type'][x]] + fail['repair_costs'][x]['pipes']
                        needed_stuff['other_stuff'] = needed_stuff['other_stuff'] +
                                                            fail['repair_costs'][x]['other_stuff']
                else:
                    needed_stuff['clamps'][fail['pipe_type']] =
                        needed_stuff['clamps'][fail['pipe_type']] + fail['repair_costs']['clamps']
                    needed_stuff['pipes'][fail['pipe_type']] =
                        needed_stuff['pipes'][fail['pipe_type']] + fail['repair_costs']['pipes']
                    needed_stuff['other_stuff'] = needed_stuff['other_stuff'] +
                                                        fail['repair_costs']['other_stuff']

        for ware in self.scenario['warehouses']:
            if ware['id'] == self.task_scheduling[team_id][-1]:
                for team4 in self.scenario['repair_teams']:
                    if team4['id'] == team_id:
                        if show_info:
                            print(f"Needed material: {needed_stuff}")
                            print(f">> State before: {team4['id'], team4['stock_levels']}")
                        # Clamps and Pipes:
                        for material in ware['data']['stock_levels']:
                            if material == 'clamps' or material == 'pipes':
                                max_material = team4['max_capacity'][material]
                                t_d1 = team4['stock_levels'][material]['D1']
                                t_d2 = team4['stock_levels'][material]['D2']
                                t_d3 = team4['stock_levels'][material]['D3']

                                if needed_stuff[material]['D1'] == 0:
                                    if t_d1 > 0:
                                        team4['stock_levels'][material]['D1'] = 0
                                        ware['data']['stock_levels'][material]['D1'] =
                                        ware['data']['stock_levels'][material]['D1'] + t_d1
                                        t_d1 = 0

                                if needed_stuff[material]['D2'] == 0:
                                    if t_d2 > 0:
                                        team4['stock_levels'][material]['D2'] = 0
                                        ware['data']['stock_levels'][material]['D2'] =
                                        ware['data']['stock_levels'][material]['D2'] + t_d2
                                        t_d2 = 0

                                if needed_stuff[material]['D3'] == 0:
                                    if t_d3 > 0:
                                        team4['stock_levels'][material]['D3'] = 0
                                        ware['data']['stock_levels'][material]['D3'] =
                                        ware['data']['stock_levels'][material]['D3'] + t_d3
                                        t_d3 = 0

                                used_space = t_d1 + t_d2 + t_d3
                                free_space = max_material - used_space

                                if needed_stuff[material]['D1'] > 0:
                                    if team4['stock_levels'][material]['D1'] <
                                                                    needed_stuff[material]['D1']:
                                        if ware['data']['stock_levels'][material]['D1'] > 0 and
                                                                                free_space > 0:
                                            if self.materials[material]['D1'] >= free_space and
                                        ware['data']['stock_levels'][material]['D1'] >= free_space:
                                                team4['stock_levels'][material]['D1'] =
                                        team4['stock_levels'][material]['D1'] + free_space
                                                ware['data']['stock_levels'][material]['D1'] =
                                        ware['data']['stock_levels'][material]['D1'] - free_space
                                                free_space = 0
                                            elif self.materials[material]['D1'] > free_space >
                                        ware['data']['stock_levels'][material]['D1']:
                                                team4['stock_levels'][material]['D1'] =
                                                team4['stock_levels'][material]['D1'] +
                                                ware['data']['stock_levels'][material]['D1']
                                                free_space = free_space -
                                                ware['data']['stock_levels'][material]['D1']
                                                ware['data']['stock_levels'][material]['D1'] = 0
```

```
                                elif self.materials[material]['D1'] < free_space and
                                     ware['data']['stock_levels'][material]['D1'] >=
                                                    self.materials[material]['D1']:
                                     team4['stock_levels'][material]['D1'] =
                                     team4['stock_levels'][material]['D1'] +
                                                    self.materials[material]['D1']
                                     ware['data']['stock_levels'][material]['D1'] =
                                      ware['data']['stock_levels'][material]['D1'] -
                                                    self.materials[material]['D1']
                                     free_space = free_space -
                                                    self.materials[material]['D1']
                                else:
                                     team4['stock_levels'][material]['D1'] =
                                     team4['stock_levels'][material]['D1'] +
                                          ware['data']['stock_levels'][material]['D1']
                                     free_space = free_space -
                                          ware['data']['stock_levels'][material]['D1']
                                     ware['data']['stock_levels'][material]['D1'] = 0

                           if needed_stuff[material]['D2'] > 0:
                               if team4['stock_levels'][material]['D2'] <
                                                        needed_stuff[material]['D2']:
                                    if ware['data']['stock_levels'][material]['D2'] > 0 and
                                                                    free_space > 0:
                                        if self.materials[material]['D2'] >= free_space and
                                    ware['data']['stock_levels'][material]['D2'] >= free_space:
                                            team4['stock_levels'][material]['D2'] =
                                    team4['stock_levels'][material]['D2'] + free_space
                                            ware['data']['stock_levels'][material]['D2'] =
                                    ware['data']['stock_levels'][material]['D2'] - free_space
                                            free_space = 0
                                        elif self.materials[material]['D2'] > free_space >
                                    ware['data']['stock_levels'][material]['D2']:
                                            team4['stock_levels'][material]['D2'] =
                                    team4['stock_levels'][material]['D2'] +
                                    ware['data']['stock_levels'][material]['D2']
                                            free_space = free_space -
                                     ware['data']['stock_levels'][material]['D2']
                                            ware['data']['stock_levels'][material]['D2'] = 0
                                        elif self.materials[material]['D2'] < free_space and
                                             ware['data']['stock_levels'][material]['D2'] >=
                                                            self.materials[material]['D2']:
                                            team4['stock_levels'][material]['D2'] =
                                     team4['stock_levels'][material]['D2'] +
                                            self.materials[material]['D2']
                                            ware['data']['stock_levels'][material]['D2'] =
                                     ware['data']['stock_levels'][material]['D2'] -
                                            self.materials[material]['D2']
                                            free_space = free_space -
                                            self.materials[material]['D2']
                                        else:
                                            team4['stock_levels'][material]['D2'] =
                                            team4['stock_levels'][material]['D2'] +
                                            ware['data']['stock_levels'][material]['D2']
                                            free_space = free_space -
                                            ware['data']['stock_levels'][material]['D2']
                                            ware['data']['stock_levels'][material]['D2'] = 0

                           if needed_stuff[material]['D3'] > 0:
                               if team4['stock_levels'][material]['D3'] <
                                                        needed_stuff[material]['D3']:
                                    if ware['data']['stock_levels'][material]['D3'] > 0 and
                                                                    free_space > 0:
                                        if self.materials[material]['D3'] >= free_space and
                                    ware['data']['stock_levels'][material]['D3'] >= free_space:
                                            team4['stock_levels'][material]['D3'] =
                                    team4['stock_levels'][material]['D3'] + free_space
                                            ware['data']['stock_levels'][material]['D3'] =
                                    ware['data']['stock_levels'][material]['D3'] - free_space
                                            free_space = 0
                                        elif self.materials[material]['D3'] > free_space >
                                    ware['data']['stock_levels'][material]['D3']:
                                            team4['stock_levels'][material]['D3'] =
                                    team4['stock_levels'][material]['D3'] +
                                    ware['data']['stock_levels'][material]['D3']
                                            free_space = free_space -
                                    ware['data']['stock_levels'][material]['D3']
                                            ware['data']['stock_levels'][material]['D3'] = 0
                                        elif self.materials[material]['D3'] < free_space and
                                    ware['data']['stock_levels'][material]['D3'] >=
                                    self.materials[material]['D3']:
                                            team4['stock_levels'][material]['D3'] =
```

```
                                    team4['stock_levels'][material]['D3'] +
                                                        self.materials[material]['D3']
                                    ware['data']['stock_levels'][material]['D3'] =
                                    ware['data']['stock_levels'][material]['D3'] -
                                                        self.materials[material]['D3']
                                    free_space = free_space -
                                                        self.materials[material]['D3']
                                else:
                                    team4['stock_levels'][material]['D3'] =
                                    team4['stock_levels'][material]['D3'] +
                                    ware['data']['stock_levels'][material]['D3']
                                    free_space = free_space -
                                    ware['data']['stock_levels'][material]['D3']
                                    ware['data']['stock_levels'][material]['D3'] = 0
                            else:
                                t_max = team4['max_capacity'][material]
                                on_board = team4['stock_levels'][material]
                                c_to_add = t_max - on_board

                                if ware['data']['stock_levels'][material] >= c_to_add:
                                    team4['stock_levels'][material] =
                                            team4['stock_levels'][material] + c_to_add
                                    ware['data']['stock_levels'][material] =
                                            ware['data']['stock_levels'][material] - c_to_add
                                else:
                                    team4['stock_levels'][material] =
                                    team4['stock_levels'][material] +
                                    ware['data']['stock_levels'][material]
                                    ware['data']['stock_levels'][material] = 0

        if show_info:
            for t in self.scenario['repair_teams']:
                if t == team_id:
                    print(f">> State after: {team_id, team4['stock_levels']}")
        return True

    def create_team_paths(self, failures, show_info=False):
        if show_info:
            print('============================== CREATE TEAM PATHS FOR FAILURES')
        team_path = dict()

        for team in self.scenario['repair_teams']:
            team_path[team['id']] = list()

        for team2 in self.scenario['repair_teams']:
            for fail in failures:
                team_path[team2['id']].append({'failure_id': fail['failure_id'], 'path':
              self.short_path_finder(self.graph, team2['localisation'], fail['start_node']['id'])})

        for team3 in team_path:
            team_path[team3] = sorted(team_path[team3], key=lambda d: d['path'][0])

        return team_path

    def check_teams_availables(self, failures, show_info=True):
        paths = self.create_team_paths(failures)
        if show_info:
            print('============================== CHECK TEAMS AVAILABLE')
            print("> Initial state:")
            for team in paths:
                print(team, paths[team])
            print('=================================================')

        for team in self.scenario['repair_teams']:
            for fail in failures:
                if show_info:
                    print(f">> Team {team['id']} try repaired {fail['failure_id']}")
                    print(f">>>> Teams material{team['stock_levels']}")
                    print(f">>>> Failure cost ({fail['pipe_type']}): {fail['repair_costs']}")

                if fail['failure_id'][0:3] == 'AG_':
                    cl = fail['repair_costs'][0]['clamps'] + fail['repair_costs'][1]['clamps']
                    pi = fail['repair_costs'][0]['pipes'] + fail['repair_costs'][1]['pipes']
                    ot = fail['repair_costs'][0]['other_stuff'] +
                                                    fail['repair_costs'][1]['other_stuff']
                    if team['stock_levels']['clamps'][fail['pipe_type'][0]] >= cl and \
                            team['stock_levels']['pipes'][fail['pipe_type'][0]] >= pi and \
                            team['stock_levels']['other_stuff'] >= ot:
                        if show_info:
                            print('============== SUCCESS\n')
                        else:
                            pass
```

```
                    else:
                        if show_info:
                            print('============== FAILED\n')

                        for x in paths[team['id']]:
                            if fail['failure_id'] == x['failure_id']:
                                paths[team['id']].remove(x)
                else:
                    if team['stock_levels']['clamps'][fail['pipe_type']] >=
                                                                fail['repair_costs']['clamps'] \
                            and team['stock_levels']['pipes'][fail['pipe_type']] >=
                                                                 fail['repair_costs']['pipes'] \
                            and team['stock_levels']['other_stuff'] >=
                                                               fail['repair_costs']['other_stuff']:
                        if show_info:
                            print('============== SUCCESS\n')
                        else:
                            pass
                    else:
                        if show_info:
                            print('============== FAILED\n')
                        for x in paths[team['id']]:
                            if fail['failure_id'] == x['failure_id']:
                                paths[team['id']].remove(x)
        if show_info:
            print("> Finish state:")
            for team in paths:
                print(team, paths[team])
        return paths

    def update_data_about_team(self, failures, team_id, failure_id, distance, show_info=False):
        for fail in failures:
            if fail['failure_id'] == failure_id:
                for team in self.scenario['repair_teams']:
                    if team['id'] == team_id:
                        if show_info:
                            print(f">> Team {team['id']} repaired {failure_id}")
                            print(f"> Teams material{team['stock_levels']}")
                            print(f"> Failure cost ({fail['pipe_type']}): {fail['repair_costs']}")

                        if fail['failure_id'][0:3] == 'AG_':
                            for i in range(0, len(fail['pipe_type'])):
                                team['stock_levels']['clamps'][fail['pipe_type'][i]] =
                                team['stock_levels']['clamps'][fail['pipe_type'][i]] -
                                fail['repair_costs'][i]['clamps']
                                team['stock_levels']['pipes'][fail['pipe_type'][i]] =
                                team['stock_levels']['pipes'][fail['pipe_type'][i]] -
                                fail['repair_costs'][i]['pipes']
                                team['stock_levels']['other_stuff'] =
                                team['stock_levels']['other_stuff'] -
                                fail['repair_costs'][i]['other_stuff']
                                team['localisation'] = fail['start_node']['id']

                                self.materials['clamps'][fail['pipe_type'][i]] =
                                self.materials['clamps'][fail['pipe_type'][i]] -
                                fail['repair_costs'][i]['clamps']
                                self.materials['pipes'][fail['pipe_type'][i]] =
                                self.materials['pipes'][fail['pipe_type'][i]] -
                                fail['repair_costs'][i]['pipes']
                                self.materials['other_stuff'] = self.materials['other_stuff'] -
                                fail['repair_costs'][i]['other_stuff']
                            team['work_time'] = round(team['work_time'] +
                                  fail['failure_time_repair'][1] + distance / 10, 0)
                        else:
                            team['stock_levels']['clamps'][fail['pipe_type']] =
                            team['stock_levels']['clamps'][fail['pipe_type']] -
                            fail['repair_costs']['clamps']
                            team['stock_levels']['pipes'][fail['pipe_type']] =
                            team['stock_levels']['pipes'][fail['pipe_type']] -
                            fail['repair_costs']['pipes']
                            team['stock_levels']['other_stuff'] =
                            team['stock_levels']['other_stuff'] -
                            fail['repair_costs']['other_stuff']
                            team['localisation'] = fail['start_node']['id']
                            self.materials['clamps'][fail['pipe_type']] =
                            self.materials['clamps'][fail['pipe_type']] -
                            fail['repair_costs']['clamps']
                            self.materials['pipes'][fail['pipe_type']] =
                            self.materials['pipes'][fail['pipe_type']] -
                            fail['repair_costs']['pipes']
                            self.materials['other_stuff'] = self.materials['other_stuff'] -
                            fail['repair_costs']['other_stuff']
```

```
                            team['work_time'] = round(team['work_time'] +
                            fail['failure_time_repair'][1] + distance / 10, 0)
                        if show_info:
                            print(f"> Teams material after work: {team['stock_levels']}\n")

    def find_min_time_team(self, show_info=False):
        min_work = {'team_id': None, 'time': float('inf')}
        for team_work in self.total_work_time:
            if show_info:
                print(f"Min time worker: {team_work, self.total_work_time[team_work]}")
            if self.total_work_time[team_work] < min_work['time']:
                min_work['team_id'] = team_work
                min_work['time'] = self.total_work_time[team_work]
            if show_info:
                print(f"Winner: {min_work}")
                print(f"Total: {self.total_work_time}")
        return min_work

    def scheduling_condition_ver_a(self, failures_to_do, fail2, show_info=True):
        self.task_scheduling[fail2['team_id']].append(fail2['failure_id'])
        self.done_failures.append(fail2['failure_id'])
        self.total_paths[fail2['team_id']].append(fail2['path'][1])
        self.update_data_about_team(failures_to_do, fail2['team_id'], fail2['failure_id'],
fail2['path'][0])
        self.reserved_teams.append(fail2['team_id'])

        for x in self.scenario['repair_teams']:
            self.number_of_task[x['id']] = len(self.task_scheduling[x['id']])
            self.total_work_time[x['id']] = x['work_time']

        for fail3 in failures_to_do:
            if fail3['failure_id'] == fail2['failure_id']:
                failures_to_do.remove(fail3)
                if show_info:
                    print(f"Deleted: {fail3['failure_id']}")

        if show_info:
            print('==== FAILURE LEFT ===')
            for f in failures_to_do:
                print(f['failure_id'])

            print('======================')
            print(f"Task scheduling: {self.task_scheduling}")
            print(f'Total work: {self.total_work_time}')
            print(f'Reserved teams: {self.reserved_teams}')
            print(f"Done failure: {self.done_failures}")
            return failures_to_do

    def level_scheduling(self, failures, show_info=True):

        failures_to_do = copy.deepcopy(failures)
        paths = self.check_teams_availables(failures_to_do, show_info=True)

        if len(failures_to_do) == 0:
            print('End!')
            return True

        if not any(paths.values()):
            if show_info:
                print('All teams to warehouse!')
            for tea in self.scenario['repair_teams']:
                self.go_to_warehouse(tea['id'], failures_to_do)
            paths = self.check_teams_availables(failures_to_do, show_info=True)

        while len(failures_to_do) > 0:
            updated_path_list = list()

            for team in paths:
                for fail in paths[team]:
                    fail['team_id'] = team
                    updated_path_list.append(fail)

            updated_path_list = sorted(updated_path_list, key=lambda d: d['path'][0])

            for fail2 in updated_path_list:
                min_worker = self.find_min_time_team()
                if show_info:
                    print(f"Analysing: {fail2['failure_id']} | min_worker:
                                                        {min_worker['team_id']}")
                    print(f"Team ID: {fail2['team_id']} \nMIN_W ID: {min_worker['team_id']} \n
                                                    DONE FAILURES: {self.done_failures}")
```

```
                    print(f"Place min worker in list: {updated_path_list.index(next(item for item
                              in updated_path_list if item['team_id'] == min_worker['team_id']))}")
                    print(f"Min worker in list: {len([next(item for item in updated_path_list if
                    item['team_id'] == min_worker['team_id'])])}")
                if len(self.reserved_teams) == self.available_repair_teams:
                    if show_info:
                        print('Start again!')
                    self.reserved_teams.clear()
                    return self.level_scheduling(failures_to_do)
                elif len([next(item for item in updated_path_list if item['team_id'] ==
                     min_worker['team_id'])]) == 1 and updated_path_list.index(next(item for item
                     in updated_path_list if item['team_id'] == min_worker['team_id'])) > 1:
                    distance_to_fail = next(item for item in updated_path_list if item['team_id']
                                                          == min_worker['team_id'])['path'][0]
                    distance_for_warehouses = {'ware_id': None, 'path': float('inf')}

                    for ware in self.scenario['warehouses']:
                        for teamx1 in self.scenario['repair_teams']:
                            if teamx1['id'] == min_worker['team_id']:
                                dist = {'ware_id': ware['id'], 'path':
                               self.short_path_finder(self.graph, ware['id'], teamx1['localisation'])}
                                if dist['path'][0] < distance_for_warehouses['path']:
                                    distance_for_warehouses['ware_id'] = ware['id']
                                    distance_for_warehouses['path'] = dist['path'][0]

                    if (distance_to_fail > distance_for_warehouses['path']) and
                                                        (distance_for_warehouses['path'] != 0):

                        self.go_to_warehouse(min_worker['team_id'], failures_to_do)
                        self.reserved_teams.clear()

                        return self.level_scheduling(failures_to_do)
                    elif (fail2['team_id'] == min_worker['team_id']) and fail2['failure_id']
                                                                    not in self.done_failures:
                        failures_to_do = self.scheduling_condition_ver_a(failures_to_do, fail2)
                        self.reserved_teams.clear()
                        return self.level_scheduling(failures_to_do)

                elif (fail2['team_id'] == min_worker['team_id']) and fail2['failure_id']
                                                                    not in self.done_failures:
                        failures_to_do = self.scheduling_condition_ver_a(failures_to_do, fail2)
                        break

    def task_scheduling_ver_a(self):
        self.scenario = self.create_scenario()
        self.needed_material()
        work_time = dict()

        for team in self.scenario['repair_teams']:
            self.task_scheduling[team['id']] = list()
            self.total_work_time[team['id']] = 0
            self.total_paths[team['id']] = list()
            work_time[team['id']] = list()
        self.level_scheduling(self.scenario['levels']['L1'])
        self.level_scheduling(self.scenario['levels']['L2'])
        self.level_scheduling(self.scenario['levels']['L3'])
        self.level_scheduling(self.scenario['levels']['L4'])
        print('========')
        print(self.task_scheduling)
        print(self.total_work_time)
        print(self.scenario['total_repair_time'])
        for team in self.total_paths:
            print(self.total_paths[team])
        print('========')
        for fail in self.scenario['levels']['L1']:
            print(fail['failure_id'], fail['failure_time_repair'])
        for fail in self.scenario['levels']['L2']:
            print(fail['failure_id'], fail['failure_time_repair'])
        for fail in self.scenario['levels']['L3']:
            print(fail['failure_id'], fail['failure_time_repair'])
        for fail in self.scenario['levels']['L4']:
            print(fail['failure_id'], fail['failure_time_repair'])

# ================================================================================ Main
new_case = Algorithm_7()
# new_case.update_data()
# new_case.create_levels()
# new_case.failure_repair_cost()
# new_case.create_scenario()
# new_case.graph_with_weights(True)
# new_case.path_finder()
# new_case.needed_material()

new_case.task_scheduling_ver_a()
```