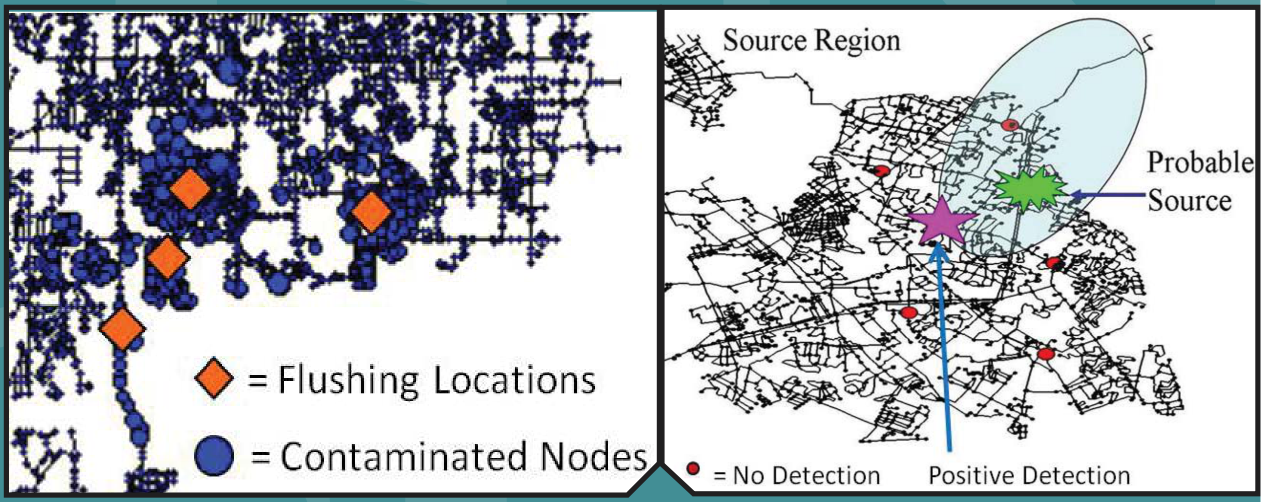
United States
Environmental Protection
Agency

EPA/600/R-20/XX | September 2020 | www.epa.gov/research

Water Security Toolkit User Manual:

Version 1.5



Office of Research and Development
National Homeland Security Research Program

Water Security Toolkit User Manual: Version 1.5

by

Katherine Klise, David Hart, John Siirola, William Hart, Cynthia Phillips
Sandia National Laboratories
Albuquerque, NM 87185

Carl Laird, Arpan Seth, Jose Santiago Rodriguez
Purdue University
West Lafayette, IN 47907

Terranna Haxton, Regan Murray, Robert Janke
U.S. Environmental Protection Agency
Cincinnati, OH 45268

Gabe Hackebeil, Angelica Mann, Shawn McGee
Texas A&M University
College Station, TX 77843

Thomas Taxon
Argonne National Laboratory
Lemont, IL 60439

Interagency Agreement DW8992450201

Terranna Haxton
U.S. Environmental Protection Agency Project Officer
Homeland Security Research Program
Cincinnati, OH 45268

Disclaimer

The U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and collaborated in the research described here under an Interagency Agreement (IA # DW8992192801 and # DW8992450201) with the Department of Energy's Sandia National Laboratories. It has been subject to the Agency's review and has been approved for publication. Note that approval does not signify that the contents necessarily reflect the views of the Agency. Mention of trade names, products, or services does not convey official EPA approval, endorsement, or recommendation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Accordingly, the United States Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so for United States Government purposes. Neither Sandia Corporation, the United States Government, nor any agency thereof, nor any of their employees makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by Sandia Corporation, the United States Government, or any agency thereof. The views and opinions expressed herein do not necessarily state or reflect those of Sandia Corporation, the United States Government or any agency thereof.

Questions concerning this document or its application should be addressed to:

Terra Haxton
Center for Environmental Solutions and Emergency Response
Office of Research and Development
U.S. Environmental Protection Agency
Cincinnati, OH 45268
Haxton.Terra@epamail.epa.gov
513-569-7810

License Notice

The Water Security Toolkit (WST) v.1.5

This software is distributed under the Revised BSD License (see below). In addition, WST leverages a variety of third-party software packages, which have separate licensing policies:

Acro	Revised BSD License
argparse	Python Software Foundation License
Boost	Boost Software License
Coopr	Revised BSD License
Coverage	BSD License
Distribute	Python Software Foundation License / Zope Public License
EPANET 2.00.12	Public Domain
EPANET-ERD	Revised BSD License
EPANET-MSX	GNU Lesser General Public License (LGPL) v.3
gcovr	Revised BSD License
GRASP	AT&T Commercial License for noncommercial use; includes `randomsample` and `sideconstraints` executable files
LZMA SDK	Public Domain
nose	GNU Lesser General Public License (LGPL) v.2.1
ordereddict	MIT License
pip	MIT License
PLY	BSD License
PyEPANET	Revised BSD License
Pyro	MIT License
PyUtilib	Revised BSD License
PyYAML	MIT License
runpy2	Python Software Foundation License
setuptools	Python Software Foundation License / Zope Public License
six	MIT License
TinyXML	zlib License
unittest2	BSD License
Utilib	Revised BSD License
virtualenv	MIT License
Vol	Common Public License
vpykit	Revised BSD License

Additionally, some precompiled WST binary distributions might bundle other third-party executables files:

Coliny	Revised BSD License (part of Acro project)
Dakota	GNU Lesser General Public License (LGPL) v.2.1
PICO	Revised BSD License (part of Acro project)

Revised BSD License

Acknowledgements

The U.S. Environmental Protection Agency (EPA) acknowledges the support in the development of the Water Security Toolkit User Manual, and in the development and testing of the Water Security Toolkit software.

The Water Security Toolkit is an extension of the Threat Ensemble Vulnerability Assessment-Sensor Placement Optimization Tool (TEVA-SPOT), which was also developed with funding from the U.S. Environmental Protection Agency through its Office of Research and Development (Interagency Agreement # DW8992192801). EPA would like to acknowledge the following individuals for their previous contributions to the development of the TEVA-SPOT toolkit software: Jonathan Berry (Sandia National Laboratories), Erik Boman (Sandia National Laboratories), Lee Ann Riesen (Sandia National Laboratories), James Uber (University of Cincinnati), and Jean-Paul Watson (Sandia National Laboratories).

Acronyms

ATUS	American Time-Use Survey
BLAS	Basic linear algebra sub-routines
CFU	Colony-forming unit
CVAR	Conditional value at risk
CWS	Contamination warning system
EA	Evolutionary algorithm
EDS	Event detection system
EPA	U.S. Environmental Protection Agency
EC	Extent of Contamination
ERD	EPANET results database file
GLPK	GNU Linear Programming Kit
GRASP	Greedy randomized adaptive sampling process
HEX	Hexadecimal
HTML	HyperText markup language
INP	EPANET input file
LP	Linear program
MC	Mass consumed
MILP	Mixed integer linear program
MIP	Mixed integer program
MSX	Multi-species extension for EPANET
NFD	Number of failed detections
NS	Number of sensors
NZD	Non-zero demand
ODE	Ordinary differential equation
PD	Population dosed
PDE	Partial differential equation
PE	Population exposed
PK	Population killed
TAI	Threat assessment input file
TCE	Tailed-conditioned expectation
TD	Time to detection
TEVA	Threat ensemble vulnerability assessment
TSB	Tryptic soy broth
TSG	Threat scenario generation file
TSI	Threat simulation input file
VAR	Value at risk
VC	Volume consumed
WST	Water Security Toolkit
YML	YAML configuration file format for WST

Symbols

Notation	Definition	Example
$\{,\}$	set brackets	$\{1, 2, 3\}$ means a set containing the values 1,2, and 3.
$\in$	is an element of	$s \in S$ means that s is an element of the set S.
$\forall$	for all	$s = 1 \; \forall \; s \in S$ means that the statement $s = 1$ is true for all s in set S.
$\sum$	summation	$\sum_{i=1}^{n} s_i$ means $s_1 + s_2 + \cdots + s_n$.
$\setminus$	set minus	$S \setminus T$ means the set that contains all those elements of S that are not in set T.
|	given	| is used to define conditional probability. $P(s\vert t)$ means the probability of s occurring given that t occurs.
|...|	cardinality	Cardinality of a set is the number of elements of the set. If set $S = \{2, 4, 6\}$, then $\vert S\vert = 3$.

Contents

1 Introduction 1

2 Getting Started 3

2.1 Obtaining the Water Security Toolkit . . . 3

2.2 Dependencies of the Water Security Toolkit . . . 3

2.3 Installing the Water Security Toolkit Binary Distributions . . . 5

2.4 Compiling the Water Security Toolkit Source Code . . . 6

2.4.1 Obtaining the Water Security Toolkit Source Code . . . 6

2.4.2 Configuring the Python Virtual Environment . . . 6

2.4.3 Building the C++ Executable Files . . . 7

2.5 Basic Usage of the Water Security Toolkit . . . 7

2.6 Verifying Installation of the Water Security Toolkit . . . 8

2.7 Uninstalling the Water Security Toolkit . . . 8

3 Contaminant Transport 9

3.1 Hydraulic and Water Quality Analysis . . . 9

3.1.1 EPANET and EPANET-MSX . . . 9

3.1.2 Merlion . . . 10

3.2 Contaminant Transport Scenarios . . . 10

3.3 `tevasim` Subcommand . . . 11

3.3.1 Configuration File . . . 11

3.3.2 Configuration Options . . . 11

3.3.3 Subcommand Output . . . 13

3.4 Contaminant Transport Examples . . . 14

3.4.1 Example 1 . . . 14

3.4.2 Example 2 . . . 15

4 Impact Assessment 17

4.1 Impact Metrics . . . 18

4.2 Human Health Impact Model . . . 20
4.2.1 Population . . . 20
4.2.2 Cumulative Dose . . . 21
4.2.3 Response . . . 22
4.2.4 Disease Progression Model . . . 22
4.3 `sim2Impact` Subcommand . . . 23
4.3.1 Configuration File . . . 23
4.3.2 Configuration Options . . . 23
4.3.3 Subcommand Output . . . 25
4.4 Impact Assessment Examples . . . 25
4.4.1 Example 1 . . . 25
4.4.2 Example 2 . . . 26
4.4.3 Example 3 . . . 26

5 Sensor Placement 28
5.1 Sensor Placement Formulations . . . 28
5.1.1 Expected-Impact Formulation . . . 29
5.1.2 Robust Formulations . . . 30
5.1.3 Side-Constrained Formulation . . . 31
5.1.4 Min-Cost Formulation . . . 32
5.2 Sensor Placement Solvers . . . 32
5.3 `sp` Subcommand . . . 33
5.3.1 Configuration File . . . 34
5.3.2 Configuration Options . . . 36
5.3.3 Subcommand Output . . . 41
5.4 Sensor Placement Examples . . . 42
5.4.1 Example 1: Solving eSP with a MIP Solver . . . 42
5.4.2 Example 2: Evaluating Solutions to eSP with Multiple Impact Files . . . 45
5.4.3 Example 3: Solving eSP with a GRASP Solver . . . 47
5.4.4 Example 4: Solving wSP with a MIP Solver . . . 48
5.4.5 Example 5: Solving cvarSP with a MIP Solver . . . 50
5.4.6 Example 6: Solving scSP with a MIP Solver . . . 51
5.4.7 Example 7: Solving mcSP with a MIP Solver . . . 56

6 Hydrant Flushing 58
6.1 Flushing Formulation . . . 59

6.2 Flushing Solvers 60
6.2.1 Evolutionary Algorithm 60
6.2.2 Network Solver 60
6.2.3 Flushing Optimization for Large Problems 61
6.2.3.1 Parallelization 61
6.2.3.2 Stop time criteria 61
6.2.3.3 Skeletonization 61
6.3 `flushing` Subcommand 62
6.3.1 Configuration File 62
6.3.2 Configuration Options 62
6.3.3 Subcommand Output 68
6.4 Flushing Response Examples 68
6.4.1 Example 1 69
6.4.2 Example 2 69
6.4.3 Example 3 69
6.4.4 Example 4 70

7 Booster Station Placement **76**
7.1 Booster Placement Using Multi-species Reaction 77
7.1.1 Booster MSX Solvers 78
7.1.1.1 Evolutionary Algorithm 78
7.1.1.2 Network Solver 79
7.1.2 `booster_msx` Subcommand 79
7.1.2.1 Configuration File 79
7.1.2.2 Configuration Options 79
7.1.2.3 Subcommand Output 85
7.2 Booster Placement Using Neutralization or Limiting Reagent Reaction 85
7.2.1 Neutralization NEUTRAL Formulation 86
7.2.2 Limiting Reagent LIMIT Formulation 86
7.2.3 Booster MIP Solvers 87
7.2.4 `booster_mip` Subcommand 88
7.2.4.1 Configuration File 88
7.2.4.2 Configuration Options 88
7.2.4.3 Subcommand Output 94
7.3 Booster Placement Subcommand Comparison 94
7.4 Booster Placement Examples 95

7.4.1 Example 1 95
7.4.2 Example 2 97

8 Source Identification **99**
8.1 Source Identification Formulations 100
8.1.1 MIP Formulations 100
8.1.2 Bayesian Probability Based Formulation 102
8.1.3 Contaminant Status Algorithm (CSA) 103
8.2 Source Identification Solvers 103
8.3 `inversion` Subcommand 104
8.3.1 Configuration File 104
8.3.2 Configuration Options 105
8.3.3 Subcommand Output 107
8.4 Source Identification Examples 108
8.4.1 Example 1 108
8.4.2 Example 2 109
8.4.3 Example 3 110

9 Uncertainty Quantification **112**
9.1 Uncertainty Quantification Method 113
9.2 `uq` Subcommand 113
9.2.1 Configuration File 113
9.2.2 Configuration Options 114
9.2.3 Subcommand Output 115
9.3 Uncertainty Quantification Example 115

10 Grab Sampling **117**
10.1 Grab Sample Formulations 117
10.1.1 Distinguishability Formulation 118
10.1.2 Probability-based Formulations 118
10.1.2.1 Maximization of expected number of scenarios that disagree with measurements 118
10.1.2.2 Maximization of scenario with least number of measurement disagreements120
10.2 Grab Sample Solvers 121
10.3 `grabsample` Subcommand 121
10.3.1 Configuration File 121
10.3.2 Configuration Options 122

10.3.3 Subcommand Output 126
10.4 Grab Sample Examples 126
10.4.1 Example 1 126
10.4.2 Example 2 128

11 Visualization 130
11.1 Color and Shape Options 131
11.2 Data from YAML Files 131
11.3 `visualization` Subcommand 132
11.3.1 Configuration File 132
11.3.2 Configuration Options 134
11.3.3 Subcommand Output 137
11.4 Visualization Examples 137
11.4.1 Example 1 137
11.4.2 Example 2 140

12 Advanced Topics and Case Studies 142
12.1 Merlion Water Quality Model 142
12.2 Average-case Sensor Placement 144
12.2.1 Computing a Bound on the Best Sensor Placement Value 144
12.2.2 Managing Sensor Placement Locations 146
12.2.3 Limited-Memory Sensor Placement Techniques 146
Scenario Aggregation: 147
Filtering Impacts: 147
Feasible Locations: 147
Witness Aggregation: 147
Skeletonization: 148
Explicit Memory Management: 148
12.2.4 Evaluating a Sensor Placement 148
12.3 Source Identification with Grab Samples Case Study 151
12.3.1 Case Study 151
12.3.2 Cycle 1 153
12.3.3 Cycle 2 154
12.3.4 Cycle 3 154
12.4 Uncertainty Reduction with Grab Samples Case Study 156
12.4.1 Case Study 156

12.4.2 Cycle 0 157
12.4.3 Cycle 1 157
12.4.4 Cycle 2 157
12.4.5 Cycle 3 158
12.5 Flushing with Source Identification Case Study 159

13 File Formats 165
13.1 Configuration File 165
13.2 Cost File 167
13.3 ERD File 168
13.4 Impact File 168
13.5 Imperfect Junction Class File 169
13.6 Imperfect Sensor Class File 169
13.7 Measurements File 169
13.8 Nodemap File 170
13.9 Scenariomap File 170
13.10 Sensor Placement File 171
13.11 TAI File 171
13.12 TSG File 173
13.13 TSI File 174
13.14 Weight File 174

14 Executable Files 176
14.1 evalsensor 176
14.1.1 Usage 176
14.1.2 Options 176
14.1.3 Arguments 177
14.2 filter_impacts 178
14.2.1 Usage 178
14.2.2 Options 178
14.2.3 Arguments 178
14.3 measuregen 179
14.3.1 Usage 179
14.3.2 Options 179
14.3.3 Arguments 180
14.4 scenarioAggr 181

14.4.1 Usage . . . 181

14.4.2 Options . . . 181

14.4.3 Arguments . . . 181

14.5 spotSkeleton . . . 182

14.5.1 Usage . . . 182

14.5.2 Arguments . . . 182

References **183**

List of Figures

2.1 The tevasim template screen output. 8

3.1 Contaminant transport simulation flowchart. 9
3.2 The tevasim configuration template file. 11
3.3 Layout of the EPANET Example Network 3. 14
3.4 Example TSG contamination scenario file. 15
3.5 The tevasim configuration file for example 1. 15
3.6 The tevasim configuration file for example 2. 15

4.1 Impact assessment flowchart. 17
4.2 The sim2Impact configuration template file. 24
4.3 The sim2Impact configuration file for example 1. 26
4.4 The sim2Impact configuration file for example 2. 26
4.5 The sim2Impact configuration file for example 3. 27

5.1 Sensor placement flowchart. 28
5.2 The sp configuration template file 34
5.3 The sp configuration template file (ctd.) 35
5.4 The sp configuration file for example 1. 43
5.5 The sp YAML output file for example 1. 43
5.6 The evalsensor output for sp example 1. 44
5.7 The sp configuration file for example 2. 45
5.8 The evalsensor output for sp example 2. 46
5.9 The sp configuration file for example 3. 47
5.10 The evalsensor output for sp example 3. 48
5.11 The sp configuration file for example 4. 49
5.12 The evalsensor output for sp example 4. 49
5.13 The sp configuration file for example 5. 50
5.14 The evalsensor output for sp example 5. 51

5.15 The sp configuration file for example 6a. 52
5.16 The evalsensor output for sp example 6a. 53
5.17 The sp configuration file for example 6b. 54
5.18 The evalsensor output for sp example 6b. 55
5.19 The sp configuration file for example 7. 56
5.20 The evalsensor output for sp example 7. 57

6.1 Flushing response simulation flowchart. 59
6.2 The flushing configuration template file. 63
6.3 The flushing configuration file for example 1. 71
6.4 The flushing YAML output file for example 1. 72
6.5 The flushing configuration file for example 2. 72
6.6 The flushing YAML output file for example 2. 73
6.7 The flushing configuration file for example 3. 73
6.8 The flushing YAML output file for example 3. 74
6.9 The flushing configuration file for example 4. 74
6.10 The flushing YAML output file for example 4. 75

7.1 Multi-species reaction booster placement flowchart. 77
7.2 MIP booster placement flowchart. 77
7.3 The booster_msx configuration template file. 80
7.4 The booster_mip configuration template file. 89
7.5 The booster_mip configuration file for example 1. 96
7.6 The booster_mip YAML output file for example 1. 97
7.7 The booster_msx configuration file for example 2. 98

8.1 Contamination source identification flowchart. 100
8.2 Three different types of contamination injection profiles. 102
8.3 The inversion configuration template file. 104
8.4 The inversion configuration file for example 1. 108
8.5 The inversion YAML output file for example 1. 109
8.6 The inversion configuration file for example 2. 110
8.7 The inversion YAML output file for example 2. 110
8.8 The inversion configuration file for example 3. 111
8.9 The inversion YAML output file for example 3. 111

9.1 Uncertainty quantification flowchart. 113

9.2 The uq configuration template file. 114
9.3 The uq configuration file for example 1. 115
9.4 List of scenarios 115
9.5 Screen output for example 1. 116

10.1 Grab sample flowchart. 117
10.2 The grabsample configuration template file. 122
10.3 The grabsample configuration file for example 1. 127
10.4 The grabsample YAML output for example 1. 128
10.5 The grabsample configuration file for example 2. 128
10.6 List of scenarios example 2. 129
10.7 List of measurements example 2. 129
10.8 The grabsample YAML output for example 2. 129

11.1 Visualization flowchart. 131
11.2 The visualization configuration template file. 133
11.3 The visualization configuration file for example 1. 138
11.4 Graphic from visualization example 1. 139
11.5 The visualization configuration file for example 2. 140
11.6 The location file used in visualization example 2. 141
11.7 Graphic from visualization example 2. 141

12.1 Illustration of the origin tracking algorithm. 143
12.2 The sp configuration file using the GLPK solver to compute a lower bound. 144
12.3 The sp YAML file with the lower bound from the GLPK solver. 145
12.4 The sp configuration file using the Lagrangian solver. 145
12.5 The sp YAML file with the lower bound from the Lagrangian solver. 146
12.6 The sp configuration file using the Lagrangian solver and the compute bound option. 147
12.7 The evalsensor example output. 149
12.8 The evalsensor output using sensor failure probabilities. 150
12.9 Illustration of the source inversion and grab sample cycling strategy. 151
12.10 Fixed sensors (blue) and contamination location (red) for case study. 152
12.11 Cycle 1 identified optimal grab sample locations (blue). 153
12.12 Cycle 2 identified optimal grab sample locations (blue). 154
12.13 The possible injection nodes (red) identified in Cycle 3. 155
12.14 Illustration of the source inversion and grab sample cycling strategy. 156
12.15 The configuration file for sampling case study. 157

12.16 EPANET Example Network 3 with grab sample locations (black diamonds), contaminated nodes (red circles), uncertain nodes (yellow circles), and clean nodes (grey circles) identified for each of the cycles in the case study. 158

12.17 Net6 water distribution network with water quality sensors. 159

12.18 Net6 with positive contamination detection at JUNCTION-1617. 160

12.19 Net6 with possible contamination sources identified by `inversion` subcommand. 161

12.20 Net6 with nodes impacted by the 25 possible contamination sources. 162

12.21 Net6 with the flushing nodes identified by the `flushing` subcommand. 163

12.22 The reduction in the PE metric for each of the 25 possible contamination sources. 164

Chapter 1

Introduction

An abundant supply of safe, high-quality drinking water is critical to modern industrialized societies. At home, water is used for drinking, cooking, washing clothes and bathing. At work, water is used to operate restaurants, hospitals and manufacturing plants. In our communities, water is used for fighting fires. Consequently, contamination of drinking water infrastructure could severely impact the public health and economic vitality of a community. The distributed physical layout of drinking water systems makes them inherently vulnerable to a variety of incidents, such as terrorist attacks, accidents and even natural disasters. The physical destruction of water infrastructure can disrupt water service to communities; specifically key facilities such as hospitals, power stations and military installations. Similarly, contamination with deadly agents could result in large numbers of illnesses and fatalities.

Since the events of September 11, 2001, water utilities have had increasing concerns about the possibility of harm to our water quality due to an accidental or intentional contamination incident within a distribution network. The U.S. EPA's Response Protocol Toolbox (EPA, 2004) provides recommendations on actions that water utilities can take to minimize potential impacts to consumers following a contamination threat or incident. Detection and consequence management are major steps in this protocol. EPA has also developed modeling and simulation tools to assist in the detection of contamination incidents in water distribution networks. The Threat Ensemble Vulnerability Assessment-Sensor Placement Optimization Tool, or TEVA-SPOT (EPA, 2011), identifies the optimal placement of online water quality monitoring sensors to detect contamination incidents. Another EPA developed tool to assist in detection is the CANARY event detection system (Hart and McKenna, 2012), which analyzes water quality data from sensors and identifies periods of anomalous water quality. These tools work together to help form a contamination warning system (CWS). The overall goal of a CWS is to detect contamination incidents in time to reduce potential public health and economic consequences. The current terminology for a CWS is a water quality surveillance and response system. For more information on CWS, see U.S. EPA Water Security Initiative (EPA, 2013b).

Should a CWS detect the presence of contamination in a water distribution network, consequence management must be employed. Decision-making tools that assist water utilities in evaluating and planning various response strategies are needed to support rapid response to contamination incidents. The Water Security Toolkit (WST) is a suite of tools that help provide the information necessary to make good decisions resulting in the minimization of further human exposure to contaminants, and the maximization of the effectiveness of intervention strategies. WST is intended to assist in:

- Planning response actions to natural disasters and terrorist attacks,
- Developing consequence management plans,
- Informing large-scale exercises/training,
- Planning response actions to address traditional utility challenges, such as pipe breaks and water quality problems and

- Evaluating implications of different response strategies.

For water utilities with hydraulic modeling expertise, WST combined with EPANET-RTX (EPA, 2013a; Hatchett et al., 2011; Janke et al., 2011) could use data from CANARY, other sensor stations and field investigations to optimize and implement response actions in real-time.

WST assists in the evaluation of multiple response actions in order to select the most beneficial consequence management strategy. It includes hydraulic and water quality modeling software and optimization methodologies to identify: (1) sensor locations to detect contamination, (2) locations in the network in which the contamination was introduced, (3) hydrants to remove contaminated water from the distribution system, (4) locations in the network to inject decontamination agents to inactivate, remove or destroy contaminants and (5) locations in the network to take grab sample to confirm contamination or cleanup.

This user manual describes the different components of WST. It is also available as a Sandia Report (Klise et al. (2015)). The manual contains one chapter on each of the water security tools:

- Contaminant transport
- Impact assessment
- Sensor placement
- Hydrant flushing
- Booster placement
- Source identification
- Grab sampling
- Visualization

Another chapter discusses advanced topics and provides case studies. WST uses YAML format configuration files to supply input parameters to each water security tool. Additional information on the YAML format can be found in File Formats Section 13.1.

The contaminant transport simulation, impact assessment and sensor placement optimization tools were all developed as part of the TEVA-SPOT Toolkit (EPA, 2011). All functionality in TEVA-SPOT has been replicated in WST using new, user friendly YAML format configuration files. WST builds upon the simulation and optimization framework of TEVA-SPOT and adds several new features. These features were all developed to model possible response action plans once a contamination incident has been detected in the system. These action plans include redirecting flow by opening hydrants and closing valves, injecting decontaminant to inactivate biological agents and using sensor measurements to identify possible source locations.

The main data requirement to use WST is a calibrated water utility network model. Additional input data is dependent on the WST application. This includes information on the simulated contamination incident(s) (e.g., type, location(s), amount), the impact metric (e.g., extent of contamination, population exposed) and the response actions (e.g., flushing hydrants, injecting disinfectant). To optimize a response action, WST must be given additional information about the potential locations for water quality sensors, hydrants to flush, valves to close, disinfectant booster stations and manual grab samples. The operating characteristics of these different response actions are also required, such as the detection limits of the water quality sensors, the rate and duration that hydrants can be flushed, the control settings for injecting disinfectant at booster stations and the number of manual grab samples that can be taken at the same time. More details on the data requirements are provided in the chapter describing each of the specific water security tool. In addition, each chapter has example applications. All examples are included with WST and can be found in the examples folder. These examples use simple networks and data files that are also distributed with WST. The examples shown in this user manual are all executed on a Linux computer, so the CPU time for each example might not be the same on computers with different operating systems.

Chapter 2

Getting Started

This chapter provides information on downloading and installing WST. WST is an open source toolkit for modeling and analyzing water distribution systems to minimize the potential impact of contamination incidents.

2.1 Obtaining the Water Security Toolkit

WST is distributed by the US EPA in both source and pre-built binary forms through the World Wide Web at `https://github.com/USEPA/Water-Security-Toolkit`. From the main WST web page, click the "Releases" link. The download page has options to download the WST source code as well as pre-built binary packages for 64-bit Windows. For most users, installing pre-built binary versions of WST is recommended.

Alternatively, the WST source code can be checked out directly from the master git version control system through `https://github.com/USEPA/Water-Security-Toolkit`. In particular, the source can be checked out with the git command:

```
git clone https://github.com/USEPA/Water-Security-Toolkit
```

2.2 Dependencies of the Water Security Toolkit

WST is a collection of Python and compiled C++ software. It has dependencies on several third-party software packages. First and foremost, a Python interpreter must be installed. WST is currently compatible with Python 2.6 or 2.7. Python 3.x is not supported. Python is available from `http://python.org/`.

The WST source code and binary distributions bundle several additional Python packages, including:

Coopr
A collection of open-source optimization-related Python packages that support a diverse set of optimization capabilities for formulating and analyzing optimization models. Coopr in turn bundles several third-party dependency libraries:

argparse
A Python command line argument parsing utility.

coverage
A Python utility for capturing and reporting code coverage.

distribute
A Python utility for building and installing Python packages.

gcovr
A utility for parsing and reporting GCOV code coverage reports.

nose
A Python test-harness driver.

ordereddict
A utility that back-ports ordered dictionaries to Python 2.6.

pip
A Python utility for installing Python packages.

ply
A general parser-lexer.

pyro
A utility for managing distributed Python execution.

runpy2
A utility that back-ports runpy functionality to Python 2.4.

setuptools
A Python utility for building and installing Python packages.

six
A utility that provides a portable interface to Python 2.x and 3.x.

unittest2
A utility that back-ports unittest functionality from Python 2.7 to 2.3-2.6.

virtualenv
A utility for creating virtual Python environments.

PyUtilib
A collection of Python utilities, including the testing harness used in WST.

PyEPANET
Python wrappers for the EPANET 2.00.12 Programmers Toolkit.

PyYAML
A YAML parser and emitter for Python.

WST subcommands can leverage numerous third-party programs that are not available in the WST zipped file that contains the source code and binary distribution:

AMPL
A commercial algebraic modeling environment, available from `http://www.ampl.com/`.

CBC
An open-source mixed-integer linear programming solver, available from `https://github.com/coin-or/Cbc`. The COIN Binary Project provides pre-compiled binaries through the CoinAll distribution, available from `https://www.coin-or.org/download/binary/CoinAll`.

CPLEX
A commercial mixed-integer linear programming solver, available from `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

Coliny
An open-source package that provides algorithms for model transformation and black-box optimization, available as part of the Dakota project at `http://dakota.sandia.gov/`.

Dakota
An open-source package that provides algorithms for black-box optimization, sensitivity analysis, surrogate modeling and uncertainty quantification, available from `http://dakota.sandia.gov/`. For Windows users, the 6.8 Windows build is recommended. The information on how to download and install Dakota is provided in the 2.3 section.

GLPK
An open-source mixed-integer linear programming solver, available from `http://www.gnu.org/software/glpk/`. Pre-compiled binary distributions are available as part of most UNIX-like operating systems. The GLPK for Windows Project provides pre-compiled Windows binaries, available from `http://winglpk.sourceforge.net/`.

Gurobi

A commercial mixed-integer linear programming solver, available from http://www.gurobi.com/.

Please refer to the individual projects' documentation for licensing, pricing and installation information.

2.3 Installing the Water Security Toolkit Binary Distributions

This is the last binary build of the WST software, since it no longer in active development. The source code is available along with the "NMAKE" files in the appropriate package directory. The build is fragile, since it might require older versions of Python and MSVS runtime libraries. Due to these limitations, the best method to compile and build WST is on a Linux machine. Associated projects under active development include the following: Water Network Tool for Resilience (WNTR) available https://github.com/USEPA/WNTR, Chama available at https://github.com/sandialabs/chama and Pecos available at https://github.com/sandialabs/pecos.

The following instructions are provided for installing WST on Windows machines. For the most up-to-date instructions, including updated third-party URLs, please see the Install page on the GitHub site (https://github.com/USEPA/Water-Security-Toolkit.

- Download Anaconda 2.7 https://www.anaconda.com/distribution/#download-section. Get the "Anaconda 2.7" version, not the Anaconda 3.x.
- Download Dakota/Coliny https://dakota.sandia.gov/downloads. Get the "Windows", "6.8", "command line only", "unsupported" version (dakota-6.8-release-public-Windows.x86.zip)
- Download CBC https://www.coin-or.org/download/binary/CoinAll Get the "Windows 1.7.4" version build date "2013-12-26 18:14" (text)
- Download WST Get the "Release 2019" from the "Releases" tab (wst-2019.zip)
- Perform the following steps:
  1. Install Anaconda2 as "Just Me" so administrator access is not needed
  2. Unzip the wst-2019.zip file into the desired directory
  3. Unzip the dakota-6.8-release-public-Windows.x86.zip file (should create a directory called the same thing, without the .zip extension, in the same directory as the zip file)
  4. Copy all the files from the dakota-6.8-release-public-Windows.x86
     bin directory into the wst-2019
     bin directory
  5. Unzip the COIN-OR-1.7.4-win-21-msvc11.zip file (should create a directory called the same thing, without the .zip extension, in the same directory as the zip file)
  6. Copy all the files from the COIN-OR-1.7.4-win-21-msvc11
     win32-msvc11
     bin directory into the wst-2019
     bin directory
  7. Open an Anaconda Prompt from the "Start -> Anaconda2" menu
  8. Change directory into the wst-2019 directory
  9. Type "install"
  10. Change directory into the wst-2019
      examples directory
  11. Run the examples (e.g., wst sp sp_ex1.yml)

2.4 Compiling the Water Security Toolkit Source Code

Compiling WST from the source code is an advanced topic and targeted only at potential developers. It assumes familiarity with compilers and build terminology. General users are strongly recommended to use the pre-built binary packages whenever possible.

Compiling WST from the source code uses the Python VirtualEnv package to set up a virtual Python environment within the WST source code distribution. The Python components of WST are installed into this virtual environment to better insulate WST from the other Python installations (and vice-versa). The compiled (C++) binary executable files are installed into a bin directory within the source code distribution. Currently, WST does not support out-of-source builds.

While WST can be compiled from the source code for Windows and Linux operating systems, Windows users are recommended to leverage the pre-built binary distributions. WST can be compiled for Linux using a 3-step process:

1. Obtain the WST source code
2. Configure the Python virtual environment
3. Build the C++ executable files

2.4.1 Obtaining the Water Security Toolkit Source Code

The WST source code can either be extracted from a downloaded source zip/tar archive or checked out directly from the repository using git. The following directions assume that the source code is in the wst-1.5 directory.

2.4.2 Configuring the Python Virtual Environment

The Python virtual environment is automatically configured by the `setup` command distributed in the top-level directory of the source code distribution:

```
cd ~/wst-1.5
./setup
```

This configures WST using the system's default Python interpreter and the bundled versions of the Python dependencies. A different version of Python can be used with WST by specifying it explicitly when running the `setup` command:

```
cd ~/wst-1.5
python2.7 ./setup
```

Setup configures the Python virtual environment within the wst/python directory (e.g.,/wst-1.5/python). The virtual interpreter and the main `wst` command both reside in wst/python/bin directory (e.g.,/wst-1.5/python/bin/wst). If only a single virtual environment is going to be on the machine, adding the wst/python/bin directory to the system PATH variable is recommended. Alternatively, the `lbin` and `lpython` commands (installed into wst/python/bin) can be used to correctly locate local binaries and the local virtual python interpreter. To run the `wst` command from anywhere under the main WST directory, use the `lbin wst` command. Similarly, to run the local python (virtual environment) interpreter, use the `lpython` command. It is safe to copy both `lbin` and `lpython` to other directories (e.g.,/bin).

After the installation of the core functionality of the python environments a couple of installations are required.

```
pip install numpy
pip install texttable
pip install matplotlib
```

2.4.3 Building the C++ Executable Files

WST relies on the GNU Autotools to manage the build process for compiled executables. In particular, Autoconf version 2.60 or newer must be installed on the system along with a relatively new C++ compiler and linker (e.g., gcc >= 3.4). The build process follows the normal `autoreconf - configure - make` sequence:

```
cd ~/wst-1.5
./setup
autoreconf -v -i -f
./configure
make
```

It is not recommended to use the `make install` command. The resulting compiled binaries reside in wst/bin, and are easily accessed from anywhere under the main WST directory using the `lbin` command.

This process could be simplified by using the main `setup` command:

```
cd ~/wst-1.5
./setup build
```

2.5 Basic Usage of the Water Security Toolkit

The main command line structure to execute a WST subcommand is the following:

```
wst SUBCOMMAND <configfile>
```

where `SUBCOMMAND` is the one of subcommands available under the `wst` command and `configfile` is the configuration file associated with the specified subcommand. The subcommands include the following:

- `tevasim`
- `sim2Impact`
- `sp`
- `flushing`
- `booster_msx`
- `booster_mip`
- `inversion`
- `grabsample`
- `visualization`

Each subcommand is described in more detail in Chapters 3 through 11.

In addition, the `--help` option prints information about the different subcommand options available.

```
wst --help
```

Each subcommand has the option to generate a template configuration file by using the following command line:

```
wst SUBCOMMAND --template <configfile>
```

where `configfile` is the name of the template configuration file created for the specified `SUBCOMMAND`.

2.6 Verifying Installation of the Water Security Toolkit

An example using one of the WST subcommands can be used to verify the proper installation of WST. This example uses the WST subcommand `tevasim`, which is documented in Chapter 3.

1. A template configuration file for the `tevasim` subcommand can be generated using the following command line, in which verify-wst.yml is the template configuration file to be created:

   ```
   wst tevasim --template verify-wst.yml
   ```

   This example assumes that the wst/bin directory was added to the PATH variable. If the path was not modified, the `wst` command would be replaced with the full path to the main WST script (e.g., C:\wst-1.5\bin\wst) in this and all subsequent commands.

2. The EPANET input file for the example network (Net3.inp) needs to be copied from the wst/examples/Net3 directory to the current working directory, since it is the network file referenced in the generated template file. On Windows (assuming WST is installed to C:\wst-1.5), the command line to copy this file is the following:

   ```
   copy C:\wst-1.5\examples\Net3\Net3.inp
   ```

   On Linux (assuming WST is installed to ~/wst-1.5), the command line to copy this file is the following:

   ```
   cp ~/wst-1.5/examples/Net3/Net3.inp
   ```

3. The `tevasim` subcommand using this example is executed with the following command line:

   ```
   wst tevasim verify-wst.yml
   ```

   This runs the `tevasim` subcommand and produces the output shown in Figure 2.1

```
WST tevasim subcommand
---------------------------
Validating configuration file
Running contaminant transport simulations

WST normal termination
---------------------------
Directory: C:/wst-1.5/examples/
Results file: Net3tevasim_output.yml
Log file: Net3tevasim_output.log
```

Figure 2.1: The `tevasim` template screen output.

2.7 Uninstalling the Water Security Toolkit

As WST does not rely on a formal installer, uninstalling WST only requires deleting the main WST directory (regardless if the pre-built binaries were installed or WST was built from the source code). If the wst/bin and/or wst/python/bin directories were added to the system PATH variable, these entries should be removed also.

Chapter 3

Contaminant Transport

This chapter describes how to simulate contamination incidents in a water distribution network, which is one of the first steps before designing a water quality sensor network or evaluating response actions to a contamination incident. The `tevasim` subcommand simulates the hydraulics and contaminant transport within a water distribution network model, which consists of pipe, node, pump, valve, storage tank and reservoir components. The `tevasim` subcommand uses the hydraulic engine from EPANET 2.00.12 to solve the flow continuity and headloss equations (Rossman, 2000). Water quality simulations are calculated using either EPANET 2.00.12 (Rossman, 2000), EPANET-MSX (Shang et al., 2011) or Merlion (Mann et al., 2012a). To increase efficiency when simulating a large ensemble of contamination incidents, the `tevasim` subcommand uses a single hydraulic simulation to simulate an ensemble of water quality simulations. A flowchart representation of the `tevasim` subcommand is shown in Figure 3.1. The utility network model is defined by an EPANET compatible network model (INP format) in WST. The simulation input is supplied through the `tevasim` WST configuration file.

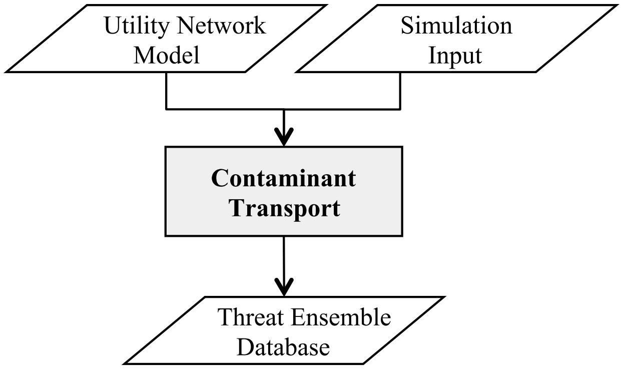


Figure 3.1: Contaminant transport simulation flowchart.

3.1 Hydraulic and Water Quality Analysis

Three water quality simulators, EPANET, EPANET-MSX and Merlion, can be used within WST. These simulators are explained in more detail in the following subsections.

3.1.1 EPANET and EPANET-MSX

EPANET performs extended-period simulation of the hydraulic and water quality behavior within pressurized pipe networks. These models can evaluate the expected flow in water distribution systems, and model the transport of contaminants and related chemical interactions. The multi-species extension, EPANET-MSX, is also included in WST to simulate contamination incidents using multi-species reactions. Any reaction dynamics between chemical and/or biological species (e.g., chemical-chemical, chemical-biological or biological-biological) can be modeled and simulated using EPANET-MSX. EPANET-MSX can be used in sensor network design (`sp` subcommand) and booster station placement (`booster_msx` subcommand). More specifics on these applications can be found in Chapters 5 and 7. Additional information on EPANET can be found at `https://www.epa.gov/water-research/epanet` and in the EPANET 2.00.12 user manual (Rossman, 2000). Additional information on EPANET-MSX can be found in the EPANET-MSX user manual (Shang et al., 2011).

3.1.2 Merlion

The `tevasim` subcommand also includes a water quality modeling framework called Merlion. Unlike EPANET, Merlion does not model bulk or wall reactions. Given hydraulic information from simulation packages like EPANET or experimental data, Merlion models the transport of a substance as it spreads through the water distribution system based on the network dynamic flow patterns. Merlion first formulates a linear water quality model with explicit all-to-all mapping (inputs include injections at all possible nodes and time steps, and outputs include concentrations at all possible nodes and time steps). This model is then used for forward tracing simulations by first specifying the injection profile and then solving the system for the network concentration profile. The linear model can also be embedded within other numerical applications or for analysis in many security applications. Using Merlion in the `tevasim` subcommand can be faster for multi-scenario simulations; however, it is also more memory intensive. Merlion can also be used to identify booster station locations, contaminant source injection locations and manual grab sample locations. More specifics about these applications are in Chapters 7, 8 and 10. More information on Merlion can be found in Section 12.1 and Mann et al. (2012a).

3.2 Contaminant Transport Scenarios

Contaminant transport scenarios can be defined directly in a WST configuration file or by using a TSI or TSG file. These options are set in the scenario block of the configuration file for all of the WST subcommands that require scenarios.

The recommended approach is to define the contamination transport scenarios directly in the scenario block of the WST configuration file. The options that must be set are the location, type, strength, species (required only for EPANET-MSX) and start and end times for the contamination scenarios. The injection location can be specified by a list of EPANET node IDs, or by the key words NZD (non-zero demand nodes) or ALL (all nodes) to create an ensemble of contaminant scenarios. The injection type can be CONCEN, MASS, FLOWPACED or SETPOINT as defined in the EPANET user manual(Rossman, 2000). CONCEN represents the concentration of an external source entering a node and applies only when the node has a net negative demand (i.e., flows into the network). MASS, FLOWPACED and SETPOINT represent booster sources, where a contaminant is injected directly into the network regardless of nodal demand. A MASS source type adds a fixed mass flow to that resulting from inflow to the node, while a FLOWPACED booster adds a fixed concentration to the resultant inflow concentration at the node. A SETPOINT booster fixes the concentration leaving the node as long as the inflow concentration was below the setpoint. The strength of a MASS source is in units of mass flow per minute, while CONCEN, FLOWPACED and SETPOINT sources are in units of concentration (mass per volume). The configuration file defines injection time in minutes and strength in mg/L or mg/min depending on the injection type.

Alternatively, the contamination transport scenarios can be defined using a TSI or TSG file. Each line of a TSI file specifies a single contaminant scenario by listing the injection location, type, species (required only for EPANET-MSX), strength and time frame. Each scenario can include multiple injection locations and multiple injection species with unique injection strengths and time frames. This format allows for the greatest flexibility in combined scenario options. For more detail on the TSI file, see File Formats Section 13.13.

The TSG file is a short hand format of the more detailed TSI file. Multiple injection locations can be specified on a single line. All permutations of the combined locations are used to create multiple scenarios. Each line of the TSG file is limited to a single injection species and time frame. For more detail on the TSG file, see File Formats Section 13.12. The TSI and TSG files specify the injection time frame in seconds (which is different than the units specified for the start and end times in the configuration file) and the strength units depend on the INP network model file units.

Specifying a TSI file overrides the TSG file, as well as the location, type, strength, species, start time and end time options specified in the WST configuration file. Specifying a TSG file overrides the location, type, strength, species, start time and end time options specified in the WST configuration file.

3.3 `tevasim` Subcommand

The `tevasim` subcommand is executed using the following command line:

```
wst tevasim <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst tevasim --help
```

3.3.1 Configuration File

The `tevasim` subcommand generates a template configuration file using the following command line:

```
wst tevasim --template <configfile>
```

The `tevasim` WST template configuration file is shown in Figure 3.2. Brief descriptions of the options are included in the template after the # sign.

```
# tevasim configuration template
network:
  epanet file: Net3.inp                  # EPANET 2.00.12 network file name
scenario:
  location: [NZD]                        # Injection location: ALL, NZD or EPANET ID
  type: MASS                             # Injection type: MASS, CONCEN, FLOWPACED, or SETPOINT
  strength: 100.0                        # Injection strength [mg/min or mg/L depending on
                                         #   type]
  species: null                          # Injection species, required for EPANET-MSX
  start time: 0                          # Injection start time [min]
  end time: 1440                         # Injection end time [min]
  tsg file: null                         # TSG file name, overrides injection parameters above
  tsi file: null                         # TSI file name, overrides TSG file
  signals: null                          # Signal files, overrides TSG or TSI files
  msx file: null                         # Multi-species extension file name
  msx species: null                      # MSX species to save
  merlion: false                         # Use Merlion as WQ simulator, true or false
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 3.2: The `tevasim` configuration template file.

3.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `tevasim` subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

scenario

location

A list that describes the injection locations for the contamination scenarios. The options are: (1) ALL, which denotes all nodes (excluding tanks and reservoirs) as contamination injection locations; (2) NZD, which denotes all nodes with non-zero demands as contamination injection locations; or (3) an EPANET node ID, which identifies a node as the contamination injection location. This allows for an easy specification of single or multiple contamination scenarios.

Required input unless a TSG or TSI file is specified.

type

The injection type for the contamination scenarios. The options are MASS, CONCEN, FLOWPACED or SETPOINT. See the EPANET 2.00.12 user manual for additional information about source types (Rossman, 2000).

Required input unless a TSG or TSI file is specified.

strength

The amount of contaminant injected into the network for the contamination scenarios. If the type option is MASS, then the units for the strength are in mg/min. If the type option is CONCEN, FLOWPACED or SETPOINT, then units are in mg/L.

Required input unless a TSG or TSI file is specified.

species

The name of the contaminant species injected into the network. This is the name of a single species. It is required when using EPANET-MSX, since multiple species might be simulated, but only one is injected into the network. For cases where multiple contaminants are injected, a TSI file must be used.

Required input for EPANET-MSX unless a TSG or TSI file is specified.

start time

The injection start time that defines when the contaminant injection begins. The time is given in minutes and is measured from the start of the simulation. For example, a value of 60 represents an injection that starts at hour 1 of the simulation.

Required input unless a TSG or TSI file is specified.

end time

The injection end time that defines when the contaminant injection stops. The time is given in minutes and is measured from the start of the simulation. For example, a value of 120 represents an injection that ends at hour 2 of the simulation.

Required input unless a TSG or TSI file is specified.

tsg file

The name of the TSG scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSG file will override the location, type, strength, species and start and end times options specified in the WST configuration file. The TSG file format is documented in File Formats Section 13.12.

Optional input.

tsi file

The name of the TSI scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSI file will override the TSG file, as well as the location, type, strength, species and start and end time options specified in the WST configuration file. The TSI file format is documented in File Formats Section 13.13.

Optional input.

signals

Name of file or directory with information to generate or load signals. If a file is provided the list of INP-TSG tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Optional input.

msx file

The name of the EPANET-MSX multi-species file that defines the multi-species reactions to be simulated using EPANET-MSX.

Required input for EPANET-MSX.

msx species

The name of the MSX species whose concentration profile will be saved by the EPANET-MSX simulation and used for later calculations.

Required input for EPANET-MSX.

merlion

A flag to indicate if the Merlion water quality simulator should be used. The options are true or false. If an MSX file is provided, EPANET-MSX will be used.

Required input, default = false.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

3.3.3 Subcommand Output

The `tevasim` subcommand creates two output files, one is in the YAML file format and the other is a log file. The YAML file is called <output prefix>tevasim_output.yml and the log file is <output prefix>tevasim_output.log. The YAML file contains the name of the EPANET report file, the name of the binary ERD database file, the run date and the CPU time. The EPANET report file and binary ERD database files are described below. The log file contains basic debugging information.

- EPANET report: This file provides information on the EPANET simulations. The EPANET report file format is described in Appendix C.3 of the EPANET 2.00.12 Users Manual (Rossman, 2000).

- ERD database: The database contains the simulation results, and is stored in four files: header file, index file, hydraulics file and a water quality file. The ERD database format is described in Geib et al. (2011). This files is not intended to be read by users but rather is read by other WST subcommands.

3.4 Contaminant Transport Examples

An EPANET network model (INP format) and a configuration file are required to run the `tevasim` subcommand.

The `tevasim` template configuration file uses the EPANET Example Network 3 (Net3.inp). The network is shown in Figure 3.3. Net3 contains 92 junctions, 2 reservoirs and 3 tanks. This network has 59 non-zero demand (NZD) nodes. The network file is setup to run a 48-hour hydraulic and water quality simulation. A 1-hour hydraulic time step and 5-minute water quality time step are used.

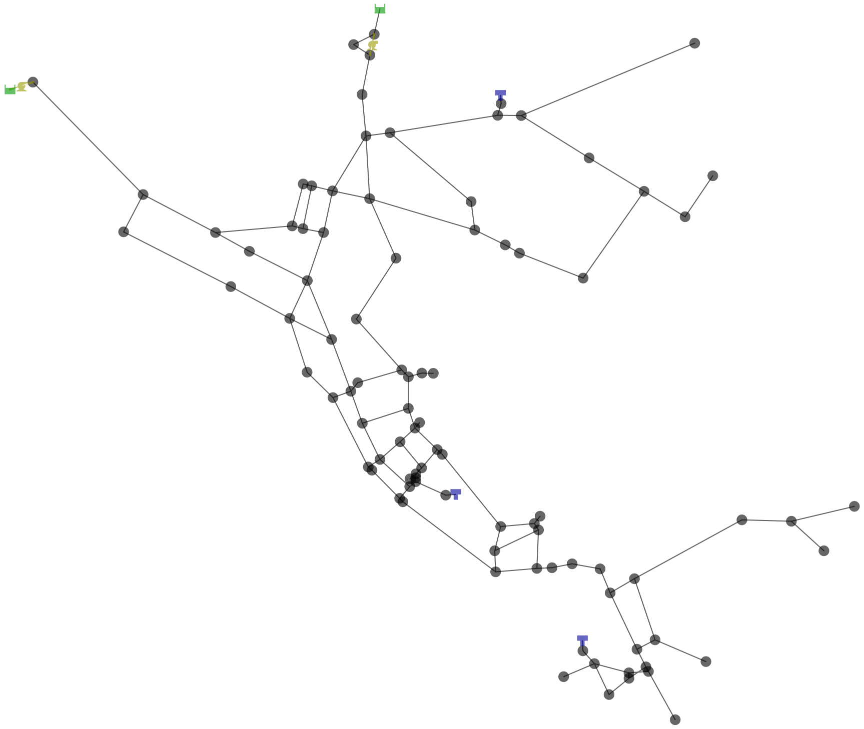


Figure 3.3: Layout of the EPANET Example Network 3.

The scenario ensemble in the `tevasim` template configuration file defines a contaminant injection from each NZD node, with a MASS injection of 100 mg/L, starting at time 0 and injecting for 24 hours (1440 minutes).

To define scenarios that start and stop at multiple times, a TSG file can be used to define the scenario set. Figure 3.4 shows the example TSG file, Net3.tsg, in which the contamination scenarios are injected at all NZD nodes starting at 12 am, 6 am, 12 pm and 6 pm for a total of 236 scenarios. Each injection lasts 24 hours and injects a contaminant at 100 mg/L.

3.4.1 Example 1

The first example uses the Net3.inp network file, the contamination scenario set defined by Net3.tsg and the Merlion water quality model. The configuration file, tevasim_ex1.yml, for this example is shown in Figure 3.5.

```
; Four 24-hour incidents (12 am, 6 am, 12 pm and 6 pm)

;Injection   Injection   Injection    Start            Stop
;Location        Type        Mass             Time (secs)      Time (secs)
NZD          MASS        100          0                86400
NZD          MASS        100          21600            108000
NZD          MASS        100          43200            129600
NZD          MASS        100          64800            151200
```

Figure 3.4: Example TSG contamination scenario file.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  tsg file: Net3/Net3.tsg
  merlion: true
configure:
  output prefix: ${CWD}/tevasim_ex1/Net3
  debug: 0
```

Figure 3.5: The `tevasim` configuration file for example 1.

The example can be executed using the following command line:

```
wst tevasim tevasim_ex1.yml
```

3.4.2 Example 2

The second example uses EPANET-MSX to simulate the transport of multiple contaminants. For a multi-species simulation, a MSX file and the MSX species must be added to the `tevasim` configuration file. The MSX species is the species whose concentration profile will be saved by EPANET-MSX to be used for future calculations. The MSX species can be different than the species which is injected into the network. The configuration file, tevasim_ex2.yml, for this example is shown in Figure 3.6. The example uses the Net3.inp network file and the MSX file, Net3_EColi_TSB.msx, which simulates the reaction dynamics between *Escherichia coli*, chlorine and a tryptic soy broth (TSB), a nutrient broth that helps to grow bacteria. For more information about this specific reaction dynamics, see the *E. coli*-TSB model described in Murray et al. (2011). In this example, both the species and MSX species are the same.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: ['15']
  type: MASS
  strength: 5.77e8
  species: EColi
  start time: 0
  end time: 360
  tsg file: null
  tsi file: null
  msx file: Net3/Net3_EColi_TSB.msx
  msx species: EColi
  merlion: false
configure:
  output prefix: ${CWD}/tevasim_ex2/Net3_EColi_TSB
  debug: 0
```

Figure 3.6: The `tevasim` configuration file for example 2.

The example can be executed using the following command line:

```
wst tevasim tevasim_ex2.yml
```

To simulate the simultaneous injection of two species, a TSI file is needed. For example, the TSI file, Net3_EColi_TSB.tsi, defines the simultaneous injection of $E. coli$ and TSB at multiple locations within Net 3. The TSI file contains explicit injections for the 59 NZD nodes used in example 1, with four species injected per node.

Chapter 4

Impact Assessment

The potential consequences of individual contamination scenarios can be quantified using the results from the contaminant transport simulations and a variety of impact assessment metrics. The `sim2Impact` subcommand performs impact assessments using the output threat ensemble database (ERD) from the `tevasim` subcommand. This analysis provides all necessary network statistics for sensor network design (described in Chapter 5) as well as response actions, such as flushing hydrants and boosting disinfectant (described in Chapters 6 and 7, respectively).

A flowchart representation of the `sim2Impact` subcommand is shown in Figure 4.1. The threat ensemble database (ERD) is the output from the `tevasim` subcommand and is required input for the `sim2Impact` subcommand. The consequences input parameters are supplied through the `sim2Impact` WST configuration file. Additional input data that describes the exposure and dose response models of a particular contaminant is required if a human health impact metric is used. These models are defined by parameters listed in a threat assessment input (TAI) file. More details on the TAI file are provided in the File Formats Section 13.11.

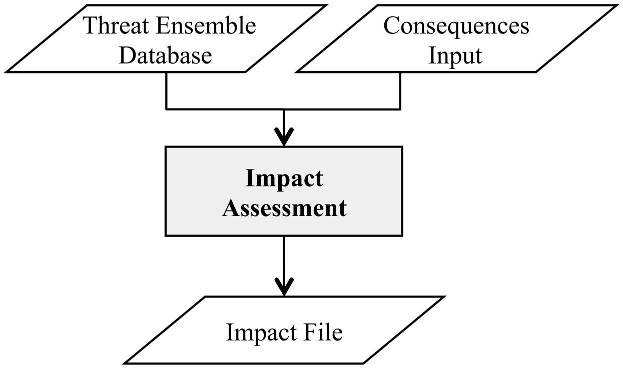


Figure 4.1: Impact assessment flowchart.

Several impact metrics are included in the `sim2Impact` subcommand to reflect different criteria that decision makers could use in sensor network design or response actions. These metrics include: population dosed (PD), population exposed (PE), population killed (PK), extent of contamination (EC), mass consumed (MC), volume consumed (VC), time to detection (TD) and number of failed detections (NFD). The equations used to compute the impact metrics are listed in Section 4.1. Impact metrics are calculated at discrete time steps for a given contamination scenario. The discrete time steps are defined by the reporting time step and the duration of the water quality simulation.

Human health impacts (PD, PE and PK) can be estimated by combining the water quality simulations with exposure models. Contaminant-specific data are needed to accurately estimate the health endpoints. For many contaminants, reliable data are lacking, and the ensuing uncertainty in the results must be understood. More information on the human health impacts is provided in Section 4.2.

4.1 Impact Metrics

Impact assessment results are calculated and stored in an impact file. This file is not typically read by a WST user, rather it is read by a sensor or response optimization routine. For each contamination scenario, the impact file contains a list of all the locations (nodes) in the network where a sensor might detect contamination from a specific scenario. Nodes that do not detect contamination are not included in the impact file for that specific scenario. For each node that detects contamination, the impact file contains the detection time and consequence at that time, as measured by one of the impact metrics.

The impact file is used as input for sensor placement optimization and during the optimization process of response actions, such as flushing hydrants and boosting disinfectant. When calculating impacts, a detection threshold can be specified such that contaminants are only detected above a specified concentration limit (the default limit is zero). Second, a response time can be specified in the `sim2Impact` configuration file, which accounts for the time needed to verify the presence of contamination (e.g., by field investigation), inform the public and/or initiate flushing, booster disinfection, or other response action (the default response time is zero). The contamination impact is computed at the time when the response has been initiated (the detection time plus response time), which is called the effective response time. Finally, a detection confidence can be defined, which specifies the number of sensors that must detect contamination from any given scenario before it is considered to be detected, at which time the impacts are calculated (the default is 1 sensor).

The impact file contains four columns of information:

- Column 1 contains the contamination scenario number, a
- Column 2 contains the node location where contamination was detected, i
- Column 3 contains the effective response time in minutes, T_i'
- Column 4 contains the impact at the effective response time as measured by a specified metric, $d_{a,i}$

The impact file is documented in the File Formats Section 13.4. The impact metric, $d_{a,i}$, is used directly in the sensor placement formulation (Equation 5.1), the flushing formulation and the booster formulation.

The effective response time at node i, T_i', is calculated using the following equation:

$$T_i' = \min\left(t : |C_{n,t} > \text{detection limit}| \geq \text{detection confidence}\right)\Delta T + \text{response time} \tag{4.1}$$

where $C_{n,t}$ is the contaminant concentration at node n at time step t for every node and time step in the water quality simulation. The concentration is typically expressed in units of milligrams per liter (mg/L). Concentration could also be a count of cells for a biological contaminant, where the units are cells/L or CFU/L (colony forming units/L). The length of the reporting time step is denoted as ΔT and has units of time. For detection, the concentration must be above the detection limit, and the number of detections must be above the detection confidence. (Note $|C_{n,t} > \text{detection limit}|$ is the number of node, time step pairs where contaminant was detected above the detection limit, this includes detection at node i).

In the impact file, the impact at the end of the simulation time is included for each contamination scenario. Essentially, this is the impact if contamination was not detected at any node location, and is often referred to as the dummy sensor location. The dummy sensor location is not a physical location in the network. For this entry, i is set to -1, T_i' is the time at the end of the water quality simulation and $d_{a,i}$ is the impact at the end of the simulation.

The impact, $d_{a,i}$, can be computed using one of the following metrics: PD, PE, PK, EC, MC, VC, TD and NFD. These metrics are defined in the following equations. In the equations, the effective response time step for node i, t'_i, equals $T'_i/\Delta T$, and subscripts n, t and p are used to reference a specific node, time step and person, respectively.

- $\mathrm{PD}_{a,i}$, population dosed, is the total number of individuals that received a cumulative dose of contaminant above a specified threshold for scenario a when contamination is detected at node i:

$$\mathrm{PD}_{a,i} = \sum_{n=1}^{N} \sum_{p=1}^{\mathrm{pop}_n} \delta_{n,p,t'_i} \text{ where } \delta_{n,p,t'_i} = \begin{cases} 1 & \text{if } d_{n,p,t'_i} > \text{ dose threshold} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

  where N is the number of nodes in the network, pop_n is the population at node n calculated using Equation 4.10, d_{n,p,t'_i} is the cumulative dose for person p at node n at the effective response time step t'_i calculated using Equation 4.14 and dose threshold is defined by the user in the TAI file.

- $\mathrm{PE}_{a,i}$, population exposed, is the number of individuals with a response to a contaminant for scenario a when contamination is detected at node i:

$$\mathrm{PE}_{a,i} = \sum_{n=1}^{N} \mathrm{pop}_n \overline{r}_{n,t'_i} \equiv \sum_{n=1}^{N} (I_{n,t'_i} + D_{n,t'_i}) \tag{4.3}$$

  where N is the number of nodes in the network, pop_n is the population at node n calculated using Equation 4.10 and $\overline{r}_{n,t'_i}$ is the percentage of the population at node n at the effective response time step t'_i that responds to a cumulative dose d_{n,p,t'_i} calculated using Equation 4.17. The variables, I_{n,t'_i} and D_{n,t'_i}, represent the number of people in the infected and diseased states, respectively, at node n at the effective response time step t'_i computed from the disease progression model described in Section 4.2.4.

- $\mathrm{PK}_{a,i}$, population killed, is the number of individuals killed by a contaminant for scenario a when contamination is detected at node i:

$$\mathrm{PK}_{a,i} = \sum_{n=1}^{N} F_{n,t'_i} \tag{4.4}$$

  where N is the number of nodes in the network and F_{n,t'_i} represents the number of people in the fatality state (number of fatalities) at node n at the effective response time step t'_i computed from the disease progression model described in Section 4.2.4.

- $\mathrm{EC}_{a,i}$, extent of contamination, is the length of contaminated pipe for scenario a when contamination is detected at node i:

$$\mathrm{EC}_{a,i} = \sum_{n=1}^{N} L_{n,t'_i} \delta_{n,t'_i} \text{ where } \delta_{n,t'_i} = \begin{cases} 1 & \text{if } C_{n,t'_i} > \text{ detection limit} \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

  where N is the number of nodes in the network, L_{n,t'_i} is the length of all pipes connected to node n with flow starting at node n at the effective response time step t'_i and C_{n,t'_i} is the contaminant concentration at node n at the effective response time step t'_i. An entire pipe is considered contaminated if the contaminant enters the pipe.

- $\mathrm{MC}_{a,i}$, mass consumed, is the cumulative mass of the contaminant consumed via the nodal demands for scenario a when contamination is detected at node i:

$$\mathrm{MC}_{a,i} = \sum_{n=1}^{N} \sum_{t=1}^{t'_i} C_{n,t} q_{n,t} \Delta T \tag{4.6}$$

where N is the number of nodes in the network, $C_{n,t}$ is the contaminant concentration at node n at time step t, $q_{n,t}$ is the demand at node n at time step t and ΔT is the length of the reporting time step. In other words, this metric measures the mass of the contaminant removed from the system at node i via nodal demand between the start of the simulation and time t'_i.

- $\mathrm{VC}_{a,i}$, volume consumed, is the cumulative volume of contaminated water consumed via nodal demand for scenario a when contamination is detected at node i:

$$\mathrm{VC}_{a,i} = \sum_{n=1}^{N} \sum_{t=1}^{t'_i} q_{n,t} \Delta T \delta_{n,t} \text{ where } \delta_{n,t} = \begin{cases} 1 & \text{if } C_{n,t} > \text{ detection limit} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

where N is the number of nodes in the network, $C_{n,t}$ is the contaminant concentration at node n at time step t, $q_{n,t}$ is the demand at node n at time step t and ΔT is the length of the reporting time step. In other words, this metric measures the volume of the contaminant removed from the system at node i via nodal demand between the start of the simulation and time t'_i.

- $\mathrm{TD}_{a,i}$, time to detection, is the time from the beginning of scenario a until contamination is first detected at a node i.

$$\mathrm{TD}_{a,i} = T'_i - \text{ injection start time} \tag{4.8}$$

where T'_i is the effective response time.

- $\mathrm{NFD}_{a,i}$, number of failed detections, is a binary value to indicate the detection of scenario a at node i:

$$\mathrm{NFD}_{a,i} = \begin{cases} 1 & \text{if scenario } a \text{ is not detected at node } i \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where the total impact is given a value of 1 if scenario a is not detected at node i or the value of 0 if scenario a is detected at node i. Since the impact file only lists nodes which detect scenarios, all node, time pairs have a total impact of 0, except for the dummy location (i = -1), which is given a value of 1.

4.2 Human Health Impact Model

The human health impact model is used to compute PD, PE and PK. In order to calculate these metrics, an estimate of the population ingesting water and the cumulative dose and response for each individual at each node is required. A disease progression model is used to compute the population susceptible, infected, diseased and killed given a cumulative dose of contaminant. Input parameters for the human health impact model are stored in a TAI file. The TAI file format is described in Section 13.11. Additional information on human health impact models can be found in the EPA compendium report (Murray et al., 2010).

4.2.1 Population

The population at each network node can either be defined explicitly in the TAI file using a population file or calculated based on the demand at each node. The population file has one line per node. Each line contains the node ID followed by the population value for that node. For the demand-based calculation, it is assumed that all water leaving the network is consumed by the population. The water consumption is more than just the ingestion of water by people, since it includes all uses of water, such as domestic, commercial, industrial, agricultural and others. Therefore, the population at node n, pop_n, is computed using the following equation:

$$\mathrm{pop}_n = \frac{\overline{q}_n}{R} \tag{4.10}$$

where $\overline{q}_n$ is the average volume of water consumed at node n per day and R is the average volume of water consumed per capita per day. The variable, R, is set in the TAI file. A USGS report provides usage rates by state and gives a nationwide average of 179 gallons per capita per day (U.S. Geological Survey, 2004). Often 200 gallons per capita per day is used for R. The population is assumed to be constant over time.

4.2.2 Cumulative Dose

At each node, the total number of people potentially ingesting water is given by pop_n. In order to compute the cumulative dose, additional information is needed, including when and how much a person drinks. Ingestion timing and volume models are used to make this calculation. Additional information on the ingestion and volume models can be found in Davis and Janke (2008). Three different ingestion timing models are available:

- Demand-based (D24): assumes that tap water is ingested at every time step in an amount proportional to the total water demand at that node.
- Fixed (F5): assumes that tap water is ingested at five fixed times during a day. These times are set to the typical starting times for the three major meals on weekdays (7:00, 12:00 and 18:00) and times halfway between these meals (9:30 and 15:00).
- Probabilistic (P5): also assumes that tap water is ingested at five times per day at major meals and halfway between them, but it uses a probabilistic approach to determine meal times. This is based on data from the American Time-Use Survey (ATUS) (Bureau of Labor Statistics and U.S. Census Bureau, 2005).

In addition, there are two ingestion volume models:

- Mean (M): assumes the same average quantity of tap water is ingested by all individuals in the population who consume tap water.
- Probabilistic (P): uses a probabilistic approach to estimate the volume ingested by individual people.

The ingestion timing model and the volume model are set in the TAI file. The D24 ingestion timing model is used only with the M volume model. Either the M or P volume models can be used with the F5 and P5 timing models. The volume models are used to determine a per capita ingestion volume, $\hat{V}_{n,p}$, in liters/day for each person p at node n. When using the M volume model, $\hat{V}_{n,p}$ is the same for each person and is commonly set to 1 to 2 liters/day. When using the P volume model, $\hat{V}_{n,p}$ can be different for each person. In each case, the volume ingested per day, $\hat{V}_{n,p}$, must be converted to a volume ingested per time step, $V_{n,p,t}$, to calculate cumulative dose for each time step.

When using the D24 ingestion timing model, $V_{n,p,t}$ is related to the demand at that node. The fraction of demand water, $rho_{n,t}$, that is ingested at node n at time step t, considering the entire length of the simulation, is computed by:

$$\rho_{n,t} = \frac{q_{n,t}}{\sum_{t=1}^{\text{nsteps}} q_{n,t}} \text{nsteps}\Delta T \tag{4.11}$$

where $q_{n,t}$ is the demand at node n at time step t, nsteps is number of time steps in the entire simulation and ΔT is the length of the reporting time step. The length of the simulation equals $\text{nsteps}\Delta T$. The volume $V_{n,p,t}$ is then computed using:

$$V_{n,p,t} = \rho_{n,t}\hat{V}_{n,p} \tag{4.12}$$

When using the F5 and P5 ingestion timing models, $\hat{V}_{n,p}$ is divided equally among each time step in which water is ingested.

$$V_{n,p,t} = \begin{cases} \hat{V}_{n,p}/5 & \text{if } t \in \{\text{ingestion time steps}\} \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

The set of ingestion time steps is calculated by dividing the ingestion times by the length of the reporting time step. This value is rounded down to the nearest discrete time step. The simulation start time should first be subtracted from the ingestion times. For example, if the set of ingestion times are defined using the F5 model as {7:00, 9:30, 12:00, 15:00 and 18:00}, the start time is 4:15 and the reporting time step is a half hour, the set of ingestion time steps are {5, 10, 15, 21 and 27}.

The cumulative dose for person p at node n at time step t, $d_{n,p,t}$, is computed using the following equation:

$$d_{n,p,t} = \sum_{j=1}^{t} C_{n,j} V_{n,p,j} \tag{4.14}$$

where $V_{n,p,j}$ is calculated using Equation 4.12 or Equation 4.13 and $C_{n,j}$ is the contaminant concentration in the water at node n at time step j as predicted by the water quality simulations. Cumulative dose is given in number of organisms or mass in milligrams.

4.2.3 Response

Dose-response functions are used to predict the percentage of the population that might experience a particular health outcome after receiving a specific cumulative dose. Two dose-response functions, $r(d_{n,p,t})$, are available in the `sim2Impact` subcommand:

- Log-Probit model:

  $$r(d_{n,p,t}) = \Phi(\beta \ln(d_{n,p,t} \mathrm{LD50})) \tag{4.15}$$

  where Φ is the cumulative distribution function of a standard normal random variable, β is related to the slope of the curve, $\mathrm{LD50}$ (or $\mathrm{ID50}$ for biological agents) is the dose at which 50% of the exposed population would die and $d_{n,p,t}$ is calculated using Equation 4.14. The parameters β and $\mathrm{LD50}$ are set in the TAI file.

- Generic logistic function:

  $$r(d_{n,p,t}) = \frac{a(1 + me^{-d_{n,p,t}/\tau})}{1 + \eta e^{-d_{n,p,t}/\tau}} \text{ where } \eta = e^{\mathrm{LD50}/\tau} - 2 \tag{4.16}$$

  where a, m, η and τ are function coefficients used to fit the model to available data and $d_{n,p,t}$ is calculated using Equation 4.14. The parameters a, m, η and τ are set in the TAI file.

The average response, $\overline{r}_{n,t}$, of the population at node n at time step t is calculated by:

$$\overline{r}_{n,t} = \frac{\sum_{p=1}^{\mathrm{pop}_n} r(d_{n,p,t})}{\mathrm{pop}_n} \tag{4.17}$$

where pop_n is calculated using Equation 4.10 and $r(d_{n,p,t})$ is calculated using Equation 4.15 or Equation 4.16.

4.2.4 Disease Progression Model

To track how the population at each node responds to a specified contaminant over time, a disease progression model is used. Given the percentage of people at each node who would become ill after being exposed to the contaminant, disease transmission models predict how the disease would progress over time. Disease models are used to predict the number of people at each node susceptible to illness from the contaminant (S), exposed to a lethal or infectious dose (I), experiencing symptoms of disease (D) and either recovering (R) or being fatally impacted (F). These equations assume that the recovered population does not rejoin the susceptible population. These quantities are predicted at each node over time according to the following differential equations:

$$\frac{dS}{dt} = -\lambda S \tag{4.18}$$

$$\frac{dI}{dt} = \lambda S - \sigma I \tag{4.19}$$

$$\frac{dD}{dt} = \sigma I - (\alpha + \nu)D \tag{4.20}$$

$$\frac{dR}{dt} = \nu D \tag{4.21}$$

$$\frac{dF}{dt} = \alpha D \tag{4.22}$$

where λ is the per capita rate of infection, σ is the per capita rate at which infected move to diseased, α is the per capita disease induced untreated death rate and ν is the per capita recovery rate, or the rate at which diseased moved to recovered or fatal states.

The infection rate, λ, is given by:

$$\lambda_{n,t} = \frac{dr_{n,p,t}}{dt}\frac{S_{n,0}}{S_{n,t}} \tag{4.23}$$

where $r(d_{n,p,t})$ is calculated using Equation 4.15 or Equation 4.16 and S is calculated by Equation 4.18.

In the TAI file, the LATENCY TIME is the inverse of σ, the FATALITY RATE is α and the FATALITY TIME is the inverse of ν. For more detail on the disease progression model and health impacts, see Murray et al. (2006).

4.3 `sim2Impact` Subcommand

The `sim2Impact` subcommand is executed using the following command line:

```
wst sim2Impact <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst sim2Impact --help
```

4.3.1 Configuration File

The `sim2Impact` subcommand generates a template configuration file using the following command line:

```
wst sim2Impact --template <configfile>
```

The `sim2Impact` WST configuration template is shown in Figure 4.2. Brief descriptions of the options are included in the configuration template after the # sign.

4.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `sim2Impact` subcommand are listed below.

impact

erd file

The name of the ERD database file that contains the contaminant transport simulation results. It is created by running the `tevasim` subcommand. Multiple ERD files (entered as a list, i.e., [<file1>, <file2>]) can be combined to generate a single impact file. This can be used to combine

```
# sim2Impact configuration template
impact:
  erd file: [Net3.erd]                  # ERD database file name
  metric: [MC]                          # Impact metric
  tai file: null                        # Health impact file name, required for public health
                                        #   metrics
  response time: 0                      # Time [min] needed to respond
  detection limit: [0.0]                # Thresholds needed to perform detection
  detection confidence: 1               # Number of sensors for detection
configure:
  output prefix: Net3                   # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                              # Debugging level, default = 0
```

Figure 4.2: The `sim2Impact` configuration template file.

simulation results from different types of contaminants, in which the ERD files were generated from different TSG files.

Required input.

metric

The impact metric used to compute the impact file. Options include EC, MC, NFD, PD, PE, PK, TD or VC. One impact file is created for each metric selected. These metrics are defined in Section 4.1.

Required input.

tai file

The name of the TAI file that contains health impact information. The TAI file format is documented in File Formats Section 13.11.

Required input if a public health metric is used (PD, PE or PK).

response time

The number of minutes that are needed to respond to the detection of a contaminant. This represents the time that it takes a water utility to stop the spread of the contaminant in the network and eliminate the consumption of contaminated water. As the response time increases, the impact increases because the contaminant affects the network for a greater length of time.

Required input, default = 0 minutes.

detection limit

The minimum concentration that must be exceeded before a sensor can detect a contaminant. There must be one threshold for each ERD file. The units of these detection limits depend on the units of the contaminant simulated for each ERD file (e.g., number of cells of a biological agent).

Required input, default = 0.

detection confidence

The number of sensors that must detect an incident before the impacts are calculated.

Required input, default = 1 sensor.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

4.3.3 Subcommand Output

The `sim2Impact` subcommand creates two output files, one is in the YAML file format and the other is a log file. The YAML file is called <output prefix>sim2Impact_output.yml and the log file is <output prefix>sim2Impact_output.log. The YAML file contains the names of the impact file(s), the ID file(s), the nodemap file and the scenario map file, as well as the run date and the CPU time. These files are described below. The log file contains basic debugging information.

- Impact file: One impact file is generated for each of the impact metrics specified. The file contains the observed impact at each location where a contamination scenario could be observed by a potential sensor. This file is not intended to be read by users, but it is used later for sensor placement or other response optimization. The impact file is documented in the File Formats Section 13.4.
- ID file: For each impact file (e.g., wst_Net3_mc.impact), a corresponding ID file is generated to map the location IDs back to the network node labels. This file is not intended to be read by users, since it is used internally by the software code.
- Nodemap file: The nodemap file maps sensor placement IDs to the network node labels (defined by EPANET). This file is not intended to be read by users, since it is used internally by the software code. The nodemap file is documented in the File Formats Section 13.8.
- Scenario map file: The scenario map file maps contamination scenario IDs to the network node labels (defined by EPANET). This file is not intended to be read by users, since it is used internally by the software code. The scenario map file is documented in the File Formats Section 13.9.

4.4 Impact Assessment Examples

After simulating the fate and transport of contaminants in a water distribution network, the output can be used to quantify the impacts of the contamination incidents. An ERD file and a configuration file are required to run the `sim2Impact` subcommand. In the following examples, the EPANET Example Network 3 is used. The output database, Net3.erd, from the first `tevasim` subcommand example is used to compute the impact assessments.

4.4.1 Example 1

Figure 4.3 shows the configuration file, sim2Impact_ex1.yml, for the first `sim2Impact` subcommand example. This example computes an impact assessment, based on Net3.erd, for the mass consumed (MC), volume consumed (VC), extent of contamination (EC), time to detection (TD), number of failed detections (NFD) and population exposed (PE) impact metrics. The TAI file, Net3_bio.tai, is added to define the human health impact for a biological contaminant. The response time, detection limit and detection confidence are all set at the default values (i.e., 0, 0, 1, respectively).

The example can be executed using the following command line:

```
wst sim2Impact sim2Impact_ex1.yml
```

For each impact metric, an impact file (e.g., Net3_pe.impact) and a corresponding ID file is generated (e.g., Net3_pe.impact.id). For each contamination scenario (shown in column 1 after a two line header), the impact file contains a list of nodes in the network (column 2) where a sensor might detect that contamination. For each such node, the impact file contains the detection time (column 3) and the total impact (column 4) given a sensor at that node is the first to detect contamination from that scenario.

```
impact:
  erd file: [Net3/Net3.erd]
  metric: [MC, VC, EC,  TD, NFD, PE]
  tai file: Net3/Net3_bio.tai
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: null
configure:
  output prefix: ${CWD}/sim2Impact_ex1/Net3
  debug: 0
```

Figure 4.3: The `sim2Impact` configuration file for example 1.

4.4.2 Example 2

The second example using the `sim2Impact` subcommand investigates the effect of changing the response time and detection limit on a specific impact metric, MC. The example 2 configuration file, sim2Impact_ex2.yml, is shown in Figure 4.4. This example uses a 60-minute response time and a detection limit of 0.1. Note that the units for detection limit are the same as for the mass values specified in the TSG file.

```
impact:
  erd file: [Net3/Net3.erd]
  metric: [MC]
  tai file: null
  response time: 60
  detection limit: [0.1]
  detection confidence: 1
  msx species: null
configure:
  output prefix: ${CWD}/sim2Impact_ex2/Net3
  debug: 0
```

Figure 4.4: The `sim2Impact` configuration file for example 2.

The example can be executed using the following command line:

```
wst sim2Impact sim2Impact_ex2.yml
```

4.4.3 Example 3

The `sim2Impact` example 3 calculates the impact associated with multi-species contamination incidents. The msx species option specifies which species concentration profile to use to calculate impact metrics. This option is required for multi-species contamination scenarios created by the `tevasim` subcommand. The configuration file for the multi-species example, sim2Impact_ex3.yml, is shown in Figure 4.5. This example uses an ERD file created by EPANET-MSX and computes the MC impact metric for the $E. coli$ species. The response time, detection limit and detection confidence are all set at their default values.

The example can be executed using the following command line:

```
wst sim2Impact sim2Impact_ex3.yml
```

```
impact:
  erd file: [Net3/Net3_EColi_TSB.erd]
  metric: [MC]
  tai file: null
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: EColi
configure:
  output prefix: ${CWD}/sim2Impact_ex3/Net3
  debug: 0
```

Figure 4.5: The `sim2Impact` configuration file for example 3.

Chapter 5

Sensor Placement

The sp subcommand optimizes the location of sensors in a water distribution network to minimize the impact of potential contamination incidents. The sp subcommand has a rich interface that supports a variety of optimization formulations, and it integrates a wide range of optimization solvers. An impact file is used to define the ensemble of contamination incidents. By default, sensors can be placed at any feasible junction in the network, but fixed and infeasible locations can be specified within the sp subcommand.

A flowchart representation of the sp subcommand is shown in Figure 5.1.

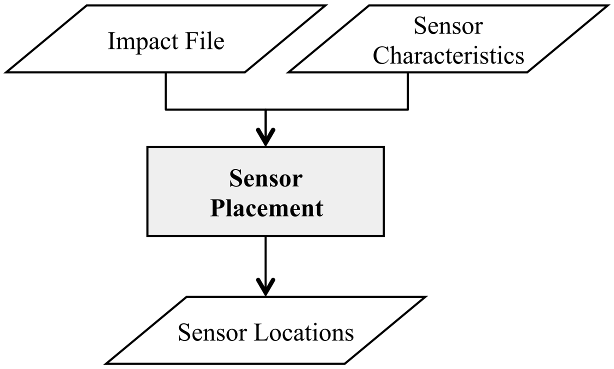


Figure 5.1: Sensor placement flowchart.

The required input for the sp subcommand is an impact file and sensor characteristics. The impact file could be created by the sim2Impact subcommand, or through some non-WST impact calculation process. Multiple impact files can be used as input to the sensor placement problem. The other required input is the sensor characteristics. These characteristics are supplied through the sp WST configuration file as well as additional files that provide details on the cost and failure rates of the sensors.

Advanced features of the sp subcommand are discussed in Section 12.2. This includes a discussion of how to specify feasible sensor locations and evaluation techniques. The section also includes advanced sensor placement methods for computing a bound on the quality of sensor networks, and techniques for minimizing memory used during sensor placement: (a) aggregation of scenarios and (b) skeletonization of the water distribution system network model.

5.1 Sensor Placement Formulations

Several sensor placement optimization formulations are available in the sp subcommand. The following formulations are described below: expected-impact, robust optimization, side-constrained and minimum cost formulations. WST also supports several sensor placement formulations that are not documented yet:

multi-stage sensor placement and imperfect sensor models.

5.1.1 Expected-Impact Formulation

The most widely studied sensor placement formulation for a contamination warning system (CWS) design is to minimize the expected impact of an ensemble of contamination incidents given a fixed number of sensors. This formulation has also become the standard formulation in the sp subcommand, because it can be effectively used to select sensor placements in large water distribution networks.

A mixed-integer programming (MIP) formulation for expected-impact sensor placement is (eSP):

$$\begin{aligned}
\text{minimize} \quad & \sum_{a \in A} \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} & & (5.1)\\
\text{subject to} \quad & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 & \forall a \in A \quad & (5.2)\\
& x_{ai} \le s_i & \forall a \in A, i \in \mathcal{L}_a \quad & (5.3)\\
& \sum_{i \in L} c_i s_i \le p & & (5.4)\\
& s_i \in \{0,1\} & \forall i \in L \quad & (5.5)\\
& 0 \le x_{ai} \le 1 & \forall a \in A, i \in \mathcal{L}_a \quad & (5.6)
\end{aligned}$$

This MIP minimizes the expected impact of a set of contamination incidents defined by A. For each incident $a \in A$, α_a is the weight of incident a, which is typically a probability. This formulation integrates contamination impact calculations, which are reported at a set of locations from the full set, denoted L, where a location refers to a network node. For each incident a, $\mathcal{L}_a \subseteq L$ is the set of locations that can be contaminated by a. Thus, a sensor at a location $i \in \mathcal{L}_a$ can detect contamination from incident a at the time contamination first arrives at location i. Each incident is witnessed by the first sensor to see it. For each incident $a \in A$ and location $i \in \mathcal{L}_a$, d_{ai} defines the impact of the contamination incident a if it is witnessed by location i. This impact metric assumes that as soon as a sensor witnesses contamination, then any further contamination impacts are mitigated (perhaps after a suitable delay that accounts for the response time of the water utility). The s_i variables indicate where sensors are placed in the network, c_i is the cost of placing a sensor at location i and p is the budget.

The x_{ai} variables indicate whether incident a is witnessed by a sensor at location i. They are defined as continuous variables between 0 and 1. In practice, there is always an optimal solution where x_{ai} is binary. If x_{ai} is fractional, then two or more equally good locations have sensors Murray et al. (2010). A given set of sensors might not be able to witness all contamination incidents. To account for this, L contains a dummy location, q. The dummy location is assigned the impact if contamination was not detected at any node location. The dummy location is not a physical location in the network. This dummy location is in all sets $\mathcal{L}_a$. If the dummy location witnesses an incident, it generally means that no real sensor can detect that incident. The impact for this location is the impact of the contamination incident after the entire contaminant transport simulation has finished, which estimates the impact that would occur without an online sensor network. The impact of a dummy detection is greater than all other impacts for each incident, so the witness variable x_{ai} for the dummy will only be selected if no sensors have been placed that can detect this incident with smaller impact.

For examples on expected-impact sensor placement (eSP), see Example 5.4.1, 5.4.2 and 5.4.3. Berry et al. (2006) describe eSP, and they note that this formulation is identical to the well-known p-median facility location problem (Mirchandani and Francis, 1990) when $c_i = 1$. In the p-median problem, p facilities (e.g., central warehouses) are to be located on m potential sites such that the sum of distances d_{ai} between each of n customers (e.g., retail outlets) and the nearest facility i is minimized. In comparing eSP and p-median problems, there is equivalence between (1) sensors and facilities, (2) contamination incidents and customers and (3) contamination impacts and distances. While eSP allows placement of at most p sensors,

p-median formulations generally enforce placement of all p facilities; in practice, the distinction is irrelevant unless p approaches the number of possible locations. This equivalence enables the application of p-median solvers to eSP (see Example 5.4.3).

5.1.2 Robust Formulations

The eSP model described in Section 5.1.1 can be viewed as optimizing one particular statistic of the distribution of impacts defined by the contaminant transport simulations. However, other statistics might provide more robust solutions that are less sensitive to changes in this distribution (Watson et al., 2006, 2009). Consider the following generalization of eSP:

$$\begin{aligned}
\text{minimize} \quad & \text{Impact}_f(\alpha, d, x) && && (5.7)\\
\text{subject to} \quad & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 && \forall a \in A && (5.8)\\
& x_{ai} \leq s_i && \forall a \in A, i \in \mathcal{L}_a && (5.9)\\
& \sum_{i \in L} c_i s_i \leq p && && (5.10)\\
& s_i \in \{0, 1\} && \forall i \in L && (5.11)\\
& 0 \leq x_{ai} \leq 1 && \forall a \in A, i \in \mathcal{L}_a && (5.12)
\end{aligned}$$

The function $\text{Impact}_f(\alpha, d, x)$ computes a statistic of the impact distribution. The following functions supported in WST have been developed by researchers to find robust solutions to optimization problems (Watson et al., 2006, 2009):

- **Mean:** This is the statistic used in eSP (Equation 5.1)
- **VaR:** Value-at-Risk (VaR) is a percentile-based metric. Given a confidence level $\beta \in (0, 1)$, the VaR is the value of the distribution at the $1 - \beta$ percentile (Topaloglou et al., 2002). The value of VaR is less than the TCE value (see below). Mathematically, suppose a random variable W describes the distribution of possible impacts. The probability that W is less than a value w is denoted as $\P[W \leq w]$. Then

  $$\text{VaR}(W, \beta) = \min\{w \mid \P[W \leq w] \geq \beta\} \qquad (5.13)$$

  Note that the distribution W changes with each sensor placement. Further, VaR can be computed using the α, d and x values.
- **TCE:** The Tail-Conditioned Expectation (TCE) is a related metric that measures the conditional expectation of impact exceeding VaR at a given confidence level. Given a confidence level $1 - \beta$, TCE is the expectation of the worst impacts whose likelihood sums to β. This value is between VaR and the worst-case value. Mathematically, then

  $$\text{TCE}(\beta) = \text{E}\left[W \mid W \geq \text{VaR}(\beta)\right] \qquad (5.14)$$
- **CVar:** The Conditional Value-at-Risk (CVaR) is a linearization of TCE investigated by Rockafellar and Uryasev (2002). CVaR approximates TCE with a continuous, piecewise-linear function of β, which enables the use of CVaR in a MIP model.
- **Worst:** The worst impact value can be easily computed, since a finite number of contamination incidents are simulated. However, this statistic is sensitive to changes in the number of contamination incidents that are simulated; adding additional contamination incidents could significantly impact this statistic.

WST includes robust MIP reformulations of eSP for the the worst and CVar statistics. The reformulation to minimize the worst contamination impact is (wSP):

$$
\begin{aligned}
\text{minimize} \quad & w && && (5.15)\\
\text{subject to} \quad & \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} \leq w && \forall a \in A && (5.16)\\
& \sum_{i \in \mathcal{L}_a} x_{ai} = 1 && \forall a \in A && (5.17)\\
& x_{ai} \leq s_i && \forall a \in A, i \in \mathcal{L}_a && (5.18)\\
& \sum_{i \in L} c_i s_i \leq p && && (5.19)\\
& s_i \in \{0,1\} && \forall i \in L && (5.20)\\
& 0 \leq x_{ai} \leq 1 && \forall a \in A, i \in \mathcal{L}_a && (5.21)
\end{aligned}
$$

This is a standard formulation for the p-center problem (Daskin, 1995; Elloumi et al., 2004).

Similarly, the reformulation to minimize CVaR is (cvarSP):

$$
\begin{aligned}
\text{minimize} \quad & v + \frac{1}{\beta} \sum_{a \in A} \alpha_a y_a && && (5.22)\\
\text{subject to} \quad & y_a \geq \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} - v && \forall a \in A && (5.23)\\
& y_q \geq 0 && \forall a \in A && (5.24)\\
& \sum_{i \in \mathcal{L}_a} x_{ai} = 1 && \forall a \in A && (5.25)\\
& x_{ai} \leq s_i && \forall a \in A, i \in \mathcal{L}_a && (5.26)\\
& \sum_{i \in L} c_i s_i \leq p && && (5.27)\\
& s_i \in \{0,1\} && \forall i \in L && (5.28)\\
& 0 \leq x_{ai} \leq 1 && \forall a \in A, i \in \mathcal{L}_a && (5.29)
\end{aligned}
$$

The variable v represents VaR, which is implicitly computed when this model is solved. See Rockafellar and Uryasev (2002) for further discussion of CVaR formulations.

Note that these formulations share a core set of constraints and variables with eSP. The difference in these models is how the objective is expressed. For examples on robust sensor placement (wSP and cvarSP), see Example 5.4.4 and 5.4.5.

5.1.3 Side-Constrained Formulation

Another natural generalization of eSP is to consider the addition of side constraints that represent bounds on alternate objectives or statistics. For example, consider a simple extension of eSP that includes a single side-constraint (scSP):

$$
\begin{aligned}
\text{minimize} \quad & \sum_{a\in A} \alpha_a \sum_{i\in\mathcal{L}_a} d_{ai} x_{ai} && && (5.30)\\
\text{subject to} \quad & \sum_{i\in\mathcal{L}_a} x_{ai} = 1 && \forall a \in A && (5.31)\\
& x_{ai} \le s_i && \forall a \in A, i \in \mathcal{L}_a && (5.32)\\
& \sum_{i\in L} c_i s_i \le p && && (5.33)\\
& s_i \in \{0,1\} && \forall i \in L && (5.34)\\
& 0 \le x_{ai} \le 1 && \forall a \in A, i \in \mathcal{L}_a && (5.35)\\
& \sum_{a\in A} \alpha_a \sum_{i\in\mathcal{L}_a} \hat{d}_{ai} x_{ai} \le G && && (5.36)
\end{aligned}
$$

The last constraint in this formulation bounds the value of an impact statistic $\hat{d}_{ai}$. Note that this statistic could easily be used as the objective for (eSP). This can be viewed as a goal constraint. Iteratively solving scSP for different goals, G, provides an assessment of the trade-off between the impact statistics in the objective and this constraint. Hence, the scSP formulation provides a mechanism for analyzing trade-offs between different objectives.

WST provides general support for side-constraints beyond what is represented in scSP, since multiple side-constraints can be specified. Additionally, robust statistics can be specified. For example, WST can express sensor placement formulations where the mean impact is minimized while the worst-case impact is constrained. See Example 5.4.6 for a demonstration of side-constrained sensor placement (scSP).

5.1.4 Min-Cost Formulation

The eSP model described in Section 5.1.1 minimizes expected contamination impact subject to a cost constraint on the number of sensors that are installed. A related sensor placement formulation is to minimize the cost of installing sensors while constraining the contamination impact to be below a specified threshold, u.

For example, eSP can be reformulated to minimize cost (mcSP):

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i\in L} c_i s_i && && (5.37)\\
\text{subject to} \quad & \sum_{i\in\mathcal{L}_a} x_{ai} = 1 && \forall a \in A && (5.38)\\
& x_{ai} \le s_i && \forall a \in A, i \in \mathcal{L}_a && (5.39)\\
& \sum_{a\in A} \alpha_a \sum_{i\in\mathcal{L}_a} d_{ai} x_{ai} \le u && && (5.40)\\
& s_i \in \{0,1\} && \forall i \in L && (5.41)\\
& 0 \le x_{ai} \le 1 && \forall a \in A, i \in \mathcal{L}_a && (5.42)
\end{aligned}
$$

See Example 5.4.7 for a demonstration of min-cost sensor placement (mcSP).

5.2 Sensor Placement Solvers

The `sp` subcommand performs optimization using a solver specified in the configuration file. All of the solvers supported by the `sp` subcommand are practical for small-sized water distribution networks, and heuristic solvers can be used to find sensor placements for very large networks.

The sp subcommand interfaces with a variety of external solvers that can be used to perform sensor placement. Several different MIP solvers can be used to find a globally optimal solution for the eSP MIP formulation. However, this might be a computationally expensive process (especially for large problems), and the size of the MIP formulation can become prohibitively large in some cases. A variety of public-domain and commercial solvers can be used by the sp subcommand, including GLPK, CBC, PICO, CPLEX, GUROBI and XPRESS.

A greedy randomized adaptive sampling process (GRASP) heuristic performs sensor placement optimization without explicitly creating a MIP formulation. Thus, this solver uses much less memory, and it usually runs very quickly. Although the GRASP heuristic does not guarantee that a globally optimal solution is found, it has proven effective at finding optimal solutions to a variety of large-scale applications. Two different implementations of the GRASP solvers can be used: an AT&T commercial solver (att_grasp) or an open-source implementation of this solver (snl_grasp).

The Lagrangian heuristic uses the structure of the p-median MIP formulation (eSP) to find near-optimal solutions while computing a lower bound on the best possible solution.

5.3 sp Subcommand

The sp subcommand is executed with the following command:

```
wst sp <configfile>
```

where configfile is a WST configuration file in the YAML format.

The --help option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst sp --help
```

Two other options can be used to print help information. The --help-problems option prints a table of the different types of optimization problems that can be solved with the sp subcommand. For example, the following is a description of the standard problem solved by the sp subcommand (eSP):

```
default, p-median, average-case perfect-sensor
---------------------------------------------
mean obj
0 constraints
perfect
single obj
1 stage
exact
```

The first row lists the different names that can be used to specify this problem type. These are synonyms, since the standard problem, eSP, is a p-median problem where the objective is an average statistic. This is also the standard perfect-sensor formulation, where all sensors are assumed to work perfectly without failures. The six rows following the dashed lines are different characteristics of this problem:

1. the type of objective used
2. the number of side-constraints
3. specify whether sensors are perfect (i.e., without failures) or whether they can fail
4. the number of objectives
5. the number of stages (i.e., time steps) in the sensor placement formulation
6. the type of solution that is required (e.g., an exact solution versus any feasible solution).

These six characteristics are used by the sp subcommand to verify the suitability of solvers that are specified for optimization.

The --help-solvers option prints a table of the different solvers that can be applied to perform optimization. For example, the following is a description of the solvers that can be used to optimize average-case perfect-sensor problems (which is the default):

```
Problem Type                  Solver        Modeling Language
======================================================================
average-case perfect-sensor   *att_grasp     none
average-case perfect-sensor   *cbc           pyomo
average-case perfect-sensor   *cplex         pyomo
average-case perfect-sensor   *glpk          pyomo
average-case perfect-sensor    gurobi        pyomo
average-case perfect-sensor   *lagrangian    none
average-case perfect-sensor    pico          ampl
average-case perfect-sensor    pico          pyomo
average-case perfect-sensor   *snl_grasp     none
average-case perfect-sensor    xpress        pyomo
```

Solvers highlighted with an asterisk are available in the current installation of WST. The modeling language indicates whether AMPL (Fourer et al., 2002), Pyomo (Hart et al., 2012) or neither is used to solve sensor placement optimization problem. Note that GLPK includes a modeling tool that includes a subset of AMPL. Thus, problems that require AMPL can be solved if either AMPL or GLPK is installed.

5.3.1 Configuration File

The sp subcommand generates a template configuration file using the following command line:

```
wst sp --template <configfile>
```

The template configuration file for the sp subcommand is shown in Figure 5.2. Brief descriptions of the options are included in the template after the # character.

```
# sp configuration template
impact data:
  -
    name: impact1                  # Impact block name
    impact file: Net3_mc.impact    # Impact file name
    nodemap file: Net3.nodemap     # Nodemap file name
    weight file: null              # Weight file name
cost:
  -
    name: null                     # Cost block name
    cost file: null                # Cost file name
objective:
  -
    name: obj1                     # Objective block name
    goal: impact1                  # Optimization objective
    statistic: MEAN                # Objective statistic
    gamma: 0.05                    # Gamma, required with statistics VAR or CVAR
constraint:
  -
    name: const1                   # Constraint block name
    goal: NS                       # Constraint goal
    statistic: TOTAL               # Constraint statistic
    gamma: 0.05                    # Gamma, required with statistics VAR or CVAR
    bound: 5                       # Constraint upper bound
```

Figure 5.2: The sp configuration template file (*continued in Figure 5.3*).

```
aggregate:
  -
    name: null                     # Aggregation block name
    type: null                     # Aggregation type: THRESHOLD, PERCENT or RATIO
    goal: null                     # Aggregation goal
    value: null                    # Aggregation value
    conserve memory: 0             # Aggregation conserve memory
    distinguish detection: 0       # Detection goal
    disable aggregation: [0]       # Aggregation disable aggregation
imperfect:
  -
    name: null                     # Imperfect block name
    sensor class file: null        # Imperfect sensor class file
    junction class file: null      # Imperfect junction class file
sensor placement:
  -
    type: default                  # Sensor placement problem type
    modeling language: NONE        # Modeling language: NONE, PYOMO or AMPL, default = NONE
    objective: obj1                # Objective block name used in sensor placement
    constraint: [const1]           # Name of constraint block(s) used in sensor placement
    imperfect: null                # Imperfect block name used in sensor placement
    aggregate: null                # Aggregate block name used in sensor placement
    compute bound: false           # Compute bounds: true or false, default = false
    presolve: true                 # Presolve problem: true or false, default = true
    compute greedy ranking: false  # Compute greedy ranking of sensor locations, default =
                                   #   false
    location:
      -
        feasible nodes: ALL     # Feasible sensor nodes
        infeasible nodes: NONE  # Infeasible sensor nodes
        fixed nodes: NONE       # Fixed sensor nodes
        unfixed nodes: NONE     # Unfixed sensor nodes
solver:
  type: snl_grasp                        # Solver type
  options:                               # A dictionary of solver options
  threads: 1                             # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 5.3: The sp configuration template file (*continued from Figure 5.2*).

5.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `sp` subcommand are listed below.

impact data

name
The name of the impact block that is used in the objective or constraint block.

Required input.

impact file
The name of the impact file that is created by `sim2Impact` and contains the detection time and the total impact given a sensor at that node is the first to detect contamination from that scenario. The impact file format is documented in File Formats Section 13.4.

Required input.

nodemap file
The name of the nodemap file that is created by `sim2Impact` and maps sensor placement ids to the network node labels. The nodemap file format is documented in File Formats Section 13.8.

Required input.

weight file
The name of the weight file that specifies the weights for contamination incidents. This file supports the optimization of weighted impact metrics. The weight file format is documented in File Formats Section 13.14.

Optional input; by default, incidents are optimized with weight 1.

cost

name
The name of the cost block that is used in the objective or constraint block.

Optional input.

cost file
The name of the cost file that contains the costs for the installation of sensors throughout the distribution network. This file contains EPANET ID/cost pairs. The cost file format is documented in File Formats Section 13.2.

Optional input.

objective

name
The name of the objective block that is used in sensor placement block.

Required input.

goal
The objective of the optimization process that defines what is going to minimized. The options are the name of the impact block, the name of the cost block, the number of sensors (NS) or the number of failed detections (NFD).

Required input.

statistic
The objective statistic. The TOTAL statistic is used when the goal is NS or NFD. When the goal is to compute a statistic of an impact block, the options are MEAN, MEDIAN, VAR, TCE, CVAR, TOTAL or WORST. For example, MEAN will minimize the mean impacts over all of the contamination scenarios, while WORST will only minimize the worst impacts from the ensemble of contamination scenarios. Required input.

gamma

The value of gamma that specifies the fraction of the distribution of impacts that will be used to compute the VAR, CVAR and TCE statistics. Gamma is assumed to be in the interval (0,1], which means that gamma can be greater than zero but less than or equal to one. It can be interpreted as specifying the 100*gamma percent of the worst contamination incidents that are used for these calculations. Required input for VAR or CVAR objective statistics, default = 0.05.

constraint

name

The name of the constraint block that is used in sensor placement block.

Required input.

goal

The constraint goal. The options are the name of the impact block name, the name of the cost block, the number of sensors (NS) or the number of failed detections (NFD).

Required input.

statistic

The constraint statistic. The TOTAL statistic is used when the goal is NS or NFD. When the goal is to compute a statistic of an impact block, the options are MEAN, MEDIAN, VAR, TCE, CVAR, TOTAL or WORST. For example, MEAN will constrain the mean impacts over all of the contamination scenarios, while WORST will only constrain the worst impacts from the ensemble of contamination scenarios.

Required input.

gamma

The value of gamma that specifies the fraction of the distribution of impacts that will be used to compute the VAR, CVAR and TCE statistics. Gamma is assumed to be in the interval (0,1], which means that gamma can be greater than zero but less than or equal to one. It can be interpreted as specifying the 100*gamma percent of the worst contamination incidents that are used for these calculations.

Required input for VAR or CVAR objective statistics, default = 0.05.

bound

The upper bound on the constraint.

Optional input.

aggregate

name

The name of the aggregation block that is used in sensor placement block.

Optional input.

type

The type of aggregation used to reduce the size of the sensor placement problem. The options are THRESHOLD, PERCENT or RATIO.

THRESHOLD is used to aggregate similar impacts by specifying a goal and a value. This is used to reduce the total size of the sensor placement formulation (for large problems). Solutions generated with non-zero thresholds are not guaranteed to be globally optimal.

PERCENT is an alternative method to compute the aggregation threshold in which the value (of the goal-value pair) is a real number between 0.0 and 1.0. Over all contamination incidents, compute the maximum difference, d, between the impact of the contamination incident if it is not detected and the impact if it is detected at the earliest possible feasible location and set the aggregation threshold to d times the aggregation percent. If both THRESHOLD and PERCENT are set to valid values, then PERCENT takes priority.

RATIO is also specified with a goal-value pair in which value is a real number between 0.0 and

1.0.

Optional input.

goal

The aggregation goal for the aggregation type.

Optional input.

value

The aggregation value for the aggregation type. If the aggregation type is PERCENT or RATIO, then this value is a real number between 0.0 and 1.0.

Optional input.

conserve memory

The maximum number of impact files that should be read into memory at any one time. This option allows impact files to be processed in a memory conserving mode if location aggregation is chosen and the original impact files are very large. For example, a conserve memory value of 10000 requests that no more than 10000 impacts should be read into memory at any one time while the original impact files are being processed into smaller aggregated files.

Optional input, default = zero to turn off this option.

distinguish detection

A goal for which aggregation should not allow incidents to become trivial. If the aggregation threshold is so large that all locations, including the dummy, would form a single superlocation, this forces the dummy to be in a superlocation by itself. Thus, the sensor placement will distinguish between detecting and not detecting. This option can be listed multiple times, to specify multiple goals.

Optional input, default = 0.

disable aggregation

Disable aggregation for this goal, even at value zero, which would incur no error. Each witness incident will be in a separate superlocation. This option can be listed multiple times to specify multiple goals. ALL can be used to specify all goals.

Optional input, default = 0.

imperfect

name

The name of the imperfect block that is used in sensor placement block.

Optional input.

sensor class file

The name of the imperfect sensor class file that defines the detection probabilities for all sensor categories. It is used with the imperfect-sensor model and must be specified in conjunction with a imperfect junction class file. The imperfect sensor class file format is documented in File Formats Section 13.6.

Optional input.

junction class file

The name of the imperfect junction class file that defines a sensor category for each network node. It is used with the imperfect-sensor model and must be specified in conjunction with a imperfect sensor class file. The imperfect junction class file format is documented in File Formats Section 13.5.

Optional input.

sensor placement

type

The sensor placement problem type. The command `wst sp --help-problems` provides a list of problem types for sensor placement. For example, average-case perfect-sensor is the standard problem type for sensor placement, since it uses the mean statistic, zero constraints, single objective and perfect sensors.

Required option, default = average-case perfect-sensor.

modeling language

The modeling language to generate the sensor placement optimization problem. The options are NONE, PYOMO or AMPL.

Required input, default = NONE.

objective

The name of the objective block previously defined to be used in sensor placement.

Required input.

constraint

The name of the constraint block previously defined to be used in sensor placement.

Required input.

imperfect

The name of the imperfect block previously defined to be used in sensor placement.

Optional input.

aggregate

The name of the aggregate block previously defined to be used in sensor placement.

Optional input.

compute bound

A flag to indicate if bounds should be computed on the sensor placement solution. The options are true or false.

Optional input, default = false.

presolve

A flag to indicate if the sensor placement problem should be presolved. The options are true or false.

Optional input, default = true.

compute greedy ranking

A flag to indicate if a greedy ranking of the sensor locations should be calculated. The options are true or false.

Optional input, default = false.

location

feasible nodes

A list that defines nodes that can be considered for the sensor placement problem. The options are: (1) ALL, which specifies all nodes as feasible sensor locations; (2) NZD, which specifies all non-zero demand nodes as feasible sensor locations; (3) a list of EPANET node IDs, which identifies specific nodes as feasible sensor locations; (4) a filename, which references a space or comma separated file containing a list of specific nodes as feasible sensor locations; or (5) NONE, which indicates this option is ignored.

Required input, default = ALL.

infeasible nodes

A list that defines nodes that cannot be considered for the sensor placement problem. The options are: (1) ALL, which specifies all nodes as infeasible sensor locations; (2) NZD, which specifies non-zero demand nodes as infeasible sensor locations; (3) a list of EPANET node IDs, which identifies specific nodes as infeasible sensor locations; (4) a filename, which references a space or comma separated file containing a list of specific nodes as infeasible sensor locations; or (5) NONE, which indicates this option is ignored.

Optional input, default = NONE.

fixed nodes

A list that defines nodes that are already sensor locations. The options are: (1) ALL, which specifies all nodes as fixed sensor locations; (2) NZD, which specifies non-zero demand nodes as fixed sensor locations; (3) a list of EPANET node IDs, which identifies specific nodes as fixed sensor locations; (4) a filename, which references a space or comma separated file containing a list of specific nodes as fixed sensor locations; or (5) NONE, which indicates this option is ignored.

Optional input, default = NONE.

unfixed nodes

A list that defines nodes that are unfixed sensor locations. The options are: (1) ALL, which specifies all nodes as unfixed sensor locations; (2) NZD, which specifies non-zero demand nodes as unfixed sensor locations; (3) a list of EPANET node IDs, which identifies specific nodes as unfixed sensor locations; (4) a filename, which references a space or comma separated file containing a list of specific nodes as unfixed sensor locations; or (5) NONE, which indicates this option is ignored.

Optional input, default = NONE.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix
The prefix used for all output files.

Required input.

output directory
The output directory to store the results.

debug
The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

5.3.3 Subcommand Output

The `sp` subcommand creates several output files. The YAML file called <output prefix>sp_output.yml contains the sensor locations, final impact metric, the run date and CPU time. For some solvers, the lower and upper bound on the objective is reported. The log file called <output prefix>sp_output.log contains basic debugging information. A visualization YAML configuration file called <output prefix>sp_output_vis.yml is also created and can be used to generate network graphics of the sensor placement solution using the `visualization` subcommand. The correct EPANET INP file must be included in the visualization YAML configuration file for the graphic to display properly.

The `sp` subcommand also outputs information a file named <output prefix>_evalsensor.out That file includes the following data:

- **Objective:** The impact metric value achieved with the sensor network design.
- **Lower bound:** The lower bound on the impact metric with the sensor network design.
- **Upper bound:** The upper bound on the impact metric with the sensor network design.
- **Solutions:** The internal node indices used by `sp` for the sensor network design.
- **Locations:** The EPANET junction labels for the sensor placement locations.
- **Sensor placement ID:** An integer ID used to distinguish the sensor network design.
- **Number of sensors:** The number of sensors in the sensor network design.
- **Total cost:** The cost of the sensor network design, which could be non-zero if cost data is provided.
- **Sensor node IDs:** The internal node indices used by `sp` for the sensor network design. The same as Solutions.
- **Sensor junctions:** The EPANET junction labels for the sensor placement locations. The same as Locations.
- **Impact file:** The name of the impact file used in the sensor network design.
- **Number of events:** The number of contamination scenarios that were simulated.

The performance of the sensor network design is summarized for each impact data file specified in the configuration file. The impact statistics included are:

- **Min impact:** The minimum impact over all contamination incidents simulated. Assuming that a sensor protects the node at which it is placed, this statistic will typically be zero.
- **Mean impact:** The mean (or average) impact over all contamination incidents simulated.
- **Lower quartile impact:** 25% of the contamination incidents, weighted by their likelihood, have an impact value less than this quartile.
- **Median impact:** 50% of the contamination incidents simulated, weighted by their likelihood, have an impact value less than this quartile.
- **Upper quartile impact:** 75% of the contamination incidents simulated, weighted by their likelihood, have an impact value less than this quartile.
- **Value at Risk (VaR):** VaR is the minimum value for which $100 \times (1-\beta)$% of the contamination incidents simulated have a smaller impact, in which β is a user-defined percentage between $0.0 < \beta < 1.0$.
- **TCE:** The tailed-conditioned expectation (TCE) is the mean value of the impacts that are greater than or equal to VaR.
- **Worst impact:** The worst impact over all contamination incidents simulated.

If the `[compute greedy ranking]` option is used, a greedy sensor placement is printed to <output prefix>_evalsensor.out. The greedy ranking places sensors one-at-a-time at the locations in the optimal sensor network design by consecutively minimizing the mean impact of placing each sensor. This analysis gives a sense of the relative priorities for these sensors. The greedy ranking is listed in terms of the sensor node IDs used by the `sp` subcommand. The corresponding EPANET node ID is listed at the top of the file.

5.4 Sensor Placement Examples

The following examples illustrate common ways that the `sp` subcommand can be used. Additional examples using the `sp` subcommand are provided in Section 12.2.

5.4.1 Example 1: Solving eSP with a MIP Solver

The first example uses the configuration file, sp_ex1.yml, shown in Figure 5.4. It specifies the impact file as Net3_ec.impact created by the `sim2Impact` subcommand using the extent of contamination (EC) impact metric and EPANET Example Network 3. The objective is to minimize the mean EC over all contamination incidents simulated while limiting the number of sensors (NS) to no more than five. The solver is the GLPK mixed-integer programming (MIP) solver, which finds globally optimal sensor placements. In addition, the greedy ranking option is used.

The `sp` subcommand is executed using the following command line:

```
wst sp sp_ex1.yml
```

The `sp` subcommand for example 1 generates the YAML output file, Net3sp_output.yml, which summarizes the sensor placement results (see Figure 5.5). The sensor network design places sensors at nodes 113, 121, 141, 163 and 209 to achieve an objective value of approximately 8655 of pipe feet contaminated.

The Net3_evalsensor.out file for the first `sp` subcommand example is shown in Figure 5.6. It displays the same sensor network design and impact value as in the YAML output file. It also includes the greedy ranking of the sensor network design, in which a sensor at node 163 would provide the greatest reduction in the impact followed by sensors at nodes 209, 113, 141 and 121. The first value in the greedy ranking is -1 (or the dummy location value), which gives the impact if no sensors are placed in the network.

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex1/Net3
  debug: 0
```

Figure 5.4: The sp configuration file for example 1.

```
# sp output
general:
  version: 1.5                                              # WST version
  date: ’2019-03-05’                                        # Run date
  cpu time: 3.83                                            # CPU time (sec)
  directory: C:/wst-1.5/examples/sp_ex1
  log file: Net3sp_output.log                               # Log file
sensor placement:
  nodes: [[’113’, ’121’, ’141’, ’163’, ’209’]]              # List of sensor nodes
  objective: [’8655.81’]                                    # Objective value
  lower bound: 8655.806356                                  # Lower bound
  upper bound: 8655.806356                                  # Upper bound
  greedy ranking: Net3_evalsensor.out                       # Upper bound
  stage 2: []                                               # Upper bound
```

Figure 5.5: The sp YAML output file for example 1.

```
------------------------------------------------------------------------------
Sensor placement id:        114124
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            16 21 28 38 65
Sensor junctions:           113 121 141 163 209

Impact File: Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8655.8064
Lower quartile impact:      0.0000
Median impact:              7110.0000
Upper quartile impact:      12444.0000
Value at Risk (VaR) (  5%): 27269.0000
TCE                 (  5%): 29853.9750
Max impact:                 36740.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
38      23998.0814
65      16138.4225
16      11534.0903
28      9821.7386
21      8655.8064
```

Figure 5.6: The evalsensor output for sp example 1.

5.4.2 Example 2: Evaluating Solutions to eSP with Multiple Impact Files

The sp subcommand can also be configured to evaluate a sensor network design using impact data not used for optimization. The second example uses the configuration file, sp_ex2.yml, shown in Figure 5.7. Several impact files, Net3_ec.impact and Net3_mc.impact, are defined in the impact block of the WST configuration file. The objective of the sensor placement optimization is to minimize the mean EC impact metric. The optimal sensor network design is then evaluated against the mass consumed (MC) impact metric.

```
impact data:
- name: ec
  impact file: ${CWD}/Net3/Net3_ec.impact
  nodemap file: ${CWD}/Net3/Net3.nodemap
- name: mc
  impact file: ${CWD}/Net3/Net3_mc.impact
  nodemap file: ${CWD}/Net3/Net3.nodemap
objective:
- name: obj1
  goal: ec
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex2/Net3
  debug: 0
```

Figure 5.7: The sp configuration file for example 2.

The sp subcommand is executed using the following command line:

```
wst sp sp_ex2.yml
```

All of the impact files specified in the configuration file are used when evaluating the sensor placement, and a greedy sensor placement is generated for each (see Figure 5.8). The sensor network design optimized for EC has a mean MC impact of 56320 mg, while the EC impact is 8655 feet. The greedy ranking of the sensors is different for the two different impact metrics. A sensor at node 209 would be the first sensor placed for the MC impact metric compared to a sensor at node 163 for the EC impact metric. The greedy ranking is a cumulative effect of adding more sensors, so the -1 value is the impact of having no sensors in the network and the next sensor location is the impact of adding one sensor and so forth down the list.

```
------------------------------------------------------------------------------
Sensor placement id:        114288
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            16 21 28 38 65
Sensor junctions:           113 121 141 163 209

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8655.8064
Lower quartile impact:      0.0000
Median impact:              7110.0000
Upper quartile impact:      12444.0000
Value at Risk (VaR) (  5%): 27269.0000
TCE                 (  5%): 29853.9750
Max impact:                 36740.0000

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                56320.3850
Lower quartile impact:      307.6690
Median impact:              4200.0900
Upper quartile impact:      137021.0000
Value at Risk (VaR) (  5%): 143999.0000
TCE                 (  5%): 143999.0000
Max impact:                 143999.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
38      23998.0814
65      16138.4225
16      11534.0903
28      9821.7386
21      8655.8064
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
------------------------------------------------------------------------------
-1      136858.7347
65      71509.1322
28      56685.0096
16      56409.8878
38      56329.1362
21      56320.3850
```

Figure 5.8: The evalsensor output for sp example 2.

5.4.3 Example 3: Solving eSP with a GRASP Solver

The third example illustrates the use of a heuristic solver for sensor placement. A GRASP heuristic iteratively applies local search to adaptively sample locations. Two GRASP heuristics are provided in WST. The AT&T GRASP solver is based on the AT&T `popstar` software, and it can be used for research purposes. The SNL GRASP solver is a new implementation of the GRASP algorithm in popstar. Example 3 uses the configuration file, sp_ex3.yml, which is shown in Figure 5.9. The sensor placement parameters are the same as in Example 5.4.1; the only difference is that the SNL GRASP solver is used instead of GLPK.

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: snl_grasp
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex3/Net3
  debug: 0
```

Figure 5.9: The `sp` configuration file for example 3.

The `sp` subcommand is executed using the following command line:

```
wst sp sp_ex3.yml
```

The Net3_evalsensor.out file for example 3 is shown in Figure 5.10. The SNL GRASP heuristic solver finds the same solution found by the GLPK solver in example 1. In addition, the solver found another solution during the search with the same performance. Since GRASP is a heuristic solver, it is not guaranteed to find sensor placements with the globally optimal value. However, GRASP has proven capable of finding optimal or near-optimal solutions even for large sensor placement problems. While the MIP solver in example 1 provided an upper and lower bound on the value of the solution, the GRASP solver does not generate these bounds since it is a heuristic.

GRASP is a heuristic search that is partly dependent on a random number generator. Consequently, users should not expect the solver to give identical results when run on different machines or operating systems. Thus, the solution shown in Figure 5.10 might not be the solution on everyone's machine.

```
--------------------------------------------------------------------------------
Sensor placement id:        115198
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            16 21 28 38 65
Sensor junctions:           113 121 141 163 209

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8655.8064
Lower quartile impact:      0.0000
Median impact:              7110.0000
Upper quartile impact:      12444.0000
Value at Risk (VaR) (  5%): 27269.0000
TCE                 (  5%): 29853.9750
Max impact:                 36740.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
--------------------------------------------------------------------------------
-1      47126.3322
38      23998.0814
65      16138.4225
16      11534.0903
28      9821.7386
21      8655.8064
```

Figure 5.10: The evalsensor output for sp example 3.

5.4.4 Example 4: Solving wSP with a MIP Solver

The subsequent examples illustrate the use of WST to solve more complex sensor placement problems. In most cases, these problems are significantly harder to solve than eSP. MIP solvers are used in the following examples to ensure consistency in the solution, but the time required to solve these problems is non-trivial.

The fourth example uses the configuration file, sp_ex4.yml, shown in Figure 5.11. The objective is to minimize the worst-case expected contamination over all contamination incidents simulated while limiting the number of sensors to no more than five. The solver is the CBC solver, which finds globally optimal sensor placements. In addition, the greedy ranking option is used.

The sp subcommand is executed using the following command line:

```
wst sp sp_ex4.yml
```

The Net3_evalsensor.out file for this example is shown in Figure 5.12. Compared with the results from example 1, this solution has a lower maximum impact for EC and a higher mean impact for EC. This reflects the difference in the objectives of these two problems.

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: WORST
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: worst-case perfect-sensor
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: cbc
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex4/Net3
  debug: 0
```

Figure 5.11: The sp configuration file for example 4.

```
------------------------------------------------------------------------------
Sensor placement id:        100087
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            15 19 24 41 66
Sensor junctions:           111 119 127 167 211

Impact File:                c:/wst-1.5/examples/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                10026.9436
Lower quartile impact:      1650.0000
Median impact:              9694.0000
Upper quartile impact:      14120.0000
Value at Risk (VaR) (  5%): 24715.0000
TCE                 (  5%): 26984.0917
Max impact:                 28290.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/examples/Net3/Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
41      24603.2763
66      17828.3250
15      14033.1216
19      10795.6131
24      10026.9436
```

Figure 5.12: The evalsensor output for sp example 4.

5.4.5 Example 5: Solving cvarSP with a MIP Solver

Example 5 uses the configuration file, sp_ex5.yml, shown in Figure 5.13. The objective is to minimize the conditional value-at-risk (CVaR) over all contamination incidents simulated while limiting the number of sensors to no more than five. The parameter $\gamma = 0.05$ specifies the weight of the tail that is used to measure CVaR (CVaR approximates the mean impact of the 5% worst scenarios). Hence, minimizing CVaR is similar to minimizing the worst-case; the difference is that minimizing CVaR reduces the impact of all of the worst 5% of the scenarios. The GLPK solver and the greedy ranking option are used.

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: CVAR
  gamma: 0.05
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5
sensor placement:
  type: robust-cvar perfect-sensor
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: cbc
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex5/Net3
  debug: 0
```

Figure 5.13: The sp configuration file for example 5.

The sp subcommand is executed using the following command line:

```
wst sp sp_ex5.yml
```

The Net3_evalsensor.out file for this example is shown in Figure 5.14. Compared with the results of example 1, this solution has a lower TCE for EC and a higher mean impact for EC. Compared with the results of example 4, this solution has the same maximum impact for EC and a lower TCE impact for EC. Both of these comparisons reflect the difference in the objectives of these problems.

```
--------------------------------------------------------------------------------
Sensor placement id:        177096
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            15 19 24 42 66
Sensor junctions:           111 119 127 169 211

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                9459.9911
Lower quartile impact:      1650.0000
Median impact:              9694.0000
Upper quartile impact:      14120.0000
Value at Risk (VaR) (  5%): 24199.0000
TCE                 (  5%): 26366.8750
Max impact:                 28290.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
--------------------------------------------------------------------------------
-1      47126.3322
42      22431.5335
66      15738.1076
15      12523.3470
19      10228.6606
24      9459.9911
```

Figure 5.14: The evalsensor output for sp example 5.

5.4.6 Example 6: Solving scSP with a MIP Solver

This example considers two sensor placement problems where a side constraint is used to limit the feasible sensor networks. Example 6a uses the configuration file, sp_ex6a.yml, shown in Figure 5.17. The objective is to minimize the mean contamination impact for EC over all contamination incidents simulated while limiting (1) the number of sensors to no more than five and (2) the mean contamination impact for MC to no more than 50000.0. The GLPK solver and the greedy ranking option are used.

The sp subcommand is executed using the following command line:

```
wst sp sp_ex6a.yml
```

The Net3_evalsensor.out file for this example is shown in Figure 5.16. By comparison with example 1, this solution has a higher mean impact for EC and a lower mean impact for MC. This comparison reflects how adding constraints to the formulation in example 1 leads to a worse solution for the objective while satisfying a side constraint.

Example 6b illustrates that a different impact statistic can be used in a side-constraint than is used in the objective. Note that the solution in Figure 5.16 has a worst-case impact statistic of 143999.0 The configuration file in Figure 5.17 uses a side-constraint where the MC impacts are constrained by the worst-case value of 150000.0.

Figure 5.18 shows the solution to Example 6b. This solution is similar to the solution in Example 6a. Although the solution has the same worst-case value for MC impacts, the mean MC impacts are larger.

```
impact data:
- name: ec
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
- name: mc
  impact file: Net3/Net3_mc.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: ec
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5
- name: const2
  goal: mc
  statistic: MEAN
  bound: 50000.0
sensor placement:
  type: side-constrained
  objective: obj1
  constraint:
    - const1
    - const2
  presolve: True
  compute greedy ranking: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex6a/Net3
  debug: 0
```

Figure 5.15: The sp configuration file for example 6a.

```
------------------------------------------------------------------------------
Sensor placement id:        180417
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            16 28 38 63 74
Sensor junctions:           113 141 163 207 237

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8763.7513
Lower quartile impact:      0.0000
Median impact:              7110.0000
Upper quartile impact:      12315.9000
Value at Risk (VaR) (  5%): 27754.8000
TCE                 (  5%): 34161.9833
Max impact:                 41105.0000

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                46060.0860
Lower quartile impact:      192.2400
Median impact:              2039.0300
Upper quartile impact:      124175.0000
Value at Risk (VaR) (  5%): 143999.0000
TCE                 (  5%): 143999.0000
Max impact:                 143999.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
63      23283.0614
38      16121.9182
16      11657.3742
28      9945.0225
74      8763.7513
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
------------------------------------------------------------------------------
-1      136858.7347
74      61640.7514
28      46758.6920
63      46281.6318
16      46085.6101
38      46060.0860
```

Figure 5.16: The evalsensor output for sp example 6a.

```
impact data:
- name: ec
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
- name: mc
  impact file: Net3/Net3_mc.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: ec
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5
- name: const2
  goal: mc
  statistic: WORST
  bound: 150000.0
sensor placement:
  type: side-constrained
  objective: obj1
  constraint:
    - const1
    - const2
  presolve: True
  compute greedy ranking: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex6b/Net3
  debug: 0
```

Figure 5.17: The sp configuration file for example 6b.

```
------------------------------------------------------------------------------
Sensor placement id:        180455
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            16 21 28 38 65
Sensor junctions:           113 121 141 163 209

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8655.8064
Lower quartile impact:      0.0000
Median impact:              7110.0000
Upper quartile impact:      12444.0000
Value at Risk (VaR) (  5%): 27269.0000
TCE                 (  5%): 29853.9750
Max impact:                 36740.0000

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                56320.3850
Lower quartile impact:      307.6690
Median impact:              4200.0900
Upper quartile impact:      137021.0000
Value at Risk (VaR) (  5%): 143999.0000
TCE                 (  5%): 143999.0000
Max impact:                 143999.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
38      23998.0814
65      16138.4225
16      11534.0903
28      9821.7386
21      8655.8064
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_mc.impact
------------------------------------------------------------------------------
-1      136858.7347
65      71509.1322
28      56685.0096
16      56409.8878
38      56329.1362
21      56320.3850
```

Figure 5.18: The evalsensor output for sp example 6b.

5.4.7 Example 7: Solving mcSP with a MIP Solver

Example 7 uses the configuration file, sp_ex7.yml, shown in Figure 5.19. The objective is to minimize the number of sensors while limiting the mean contamination impact for EC over all contamination incidents to no more than 5000.0. The CBC solver and the greedy ranking option are used.

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: NS
  statistic: TOTAL
constraint:
- name: const1
  goal: impact1
  statistic: MEAN
  bound: 5000.0
sensor placement:
  type: min-sensors
  objective: obj1
  constraint: const1
  presolve: True
  compute greedy ranking: True
solver:
  type: cbc
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/sp_ex7/Net3
  debug: 0
```

Figure 5.19: The sp configuration file for example 7.

The sp subcommand is executed using the following command line:

```
wst sp sp_ex7.yml
```

The Net3_evalsensor.out file for this example is shown in Figure 5.20. By comparison with example 1, this solution uses 11 sensors to find a lower mean impact for EC. Both the mcSP and eSP formulations can be used to explore the trade-off between number of sensors and contamination impact. However, the eSP formulation is much easier to solve, especially for large-scale sensor placement problems.

```
------------------------------------------------------------------------------
Sensor placement id:        180497
Number of sensors:          11
Total cost:                 0
Sensor node IDs:            10 16 18 21 24 32 38 46 53 63 74
Sensor junctions:           101 113 117 121 127 149 163 179 191 207 237

Impact File:                c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                4724.7551
Lower quartile impact:      0.0000
Median impact:              3695.0000
Upper quartile impact:      9485.0000
Value at Risk (VaR) (  5%): 12230.0000
TCE                 (  5%): 14060.8333
Max impact:                 18020.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors: c:/wst-1.5/doc/wst/examples/sp/Net3/Net3_ec.impact
------------------------------------------------------------------------------
-1      47126.3322
63      23283.0614
38      16121.9182
16      11657.3742
21      10174.8530
32      8824.5564
74      7643.2852
10      6916.9716
46      6255.5941
24      5677.5178
53      5190.0517
18      4724.7551
```

Figure 5.20: The evalsensor output for sp example 7.

Chapter 6

Hydrant Flushing

A common operational approach that water utilities use to address water quality concerns is flushing, which is the purging of water from the distribution network via a fire hydrant or blow-off port. Many utilities flush water mains following maintenance work or in response to customer complaints. Flushing can remove the sources of poor water quality (e.g., pipe corrosion, bio-film microorganisms), as well as loose or suspended material that has accumulated in low-flow portions or dead-ends of the distribution system. It is a response option that can be undertaken relatively quickly after a contamination incident, and it can be made more efficient through the careful selection of where to implement flushing activities. This chapter describes the `flushing` subcommand in WST that assists in the identification of effective hydrant locations to flush in order to remove contaminated water and the valves to close in order to direct the contaminated water towards the hydrants.

A flowchart representation of the `flushing` subcommand is shown in Figure 6.1. The `flushing` subcommand employs an iterative process that combines contaminant transport, impact assessment and optimization. The optimization process identifies a set of flushing activities that are simulated in the contaminant transport process and evaluated based upon the impact assessment process. Since the `flushing` subcommand relies on the `tevasim` and `sim2Impact` subcommands, their required input is also required for the `flushing` subcommand. In addition, the sensor network design used to detect the contamination incident(s) and the flushing characteristics are required inputs. The utility network model is defined by a EPANET 2.00.12 INP file, while the rest of the input can be specified in the `flushing` WST configuration file.

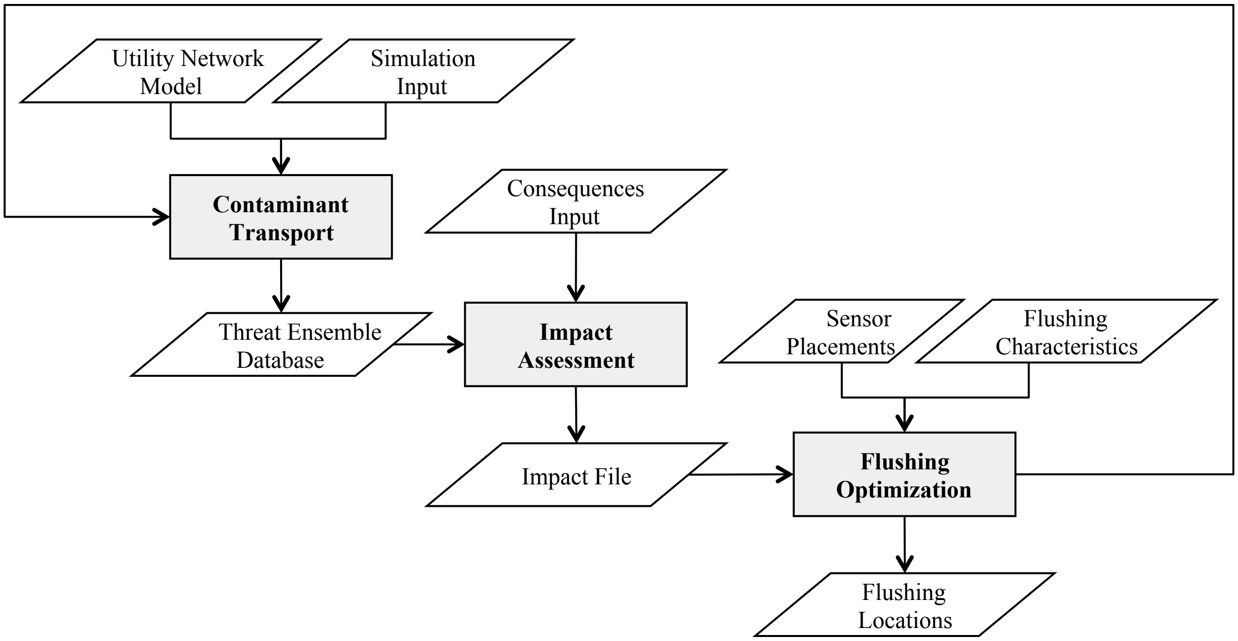


Figure 6.1: Flushing response simulation flowchart.

6.1 Flushing Formulation

The flushing problem formulation can be summarized as selecting a set of hydrant locations to flush and valves to close that minimizes the average impact of all contamination incidents given a set of potential hydrant locations to flush and valves to close. The mathematical formulation can be written as follows:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{A}\sum_{a\in A} d_{a,\max} && && (6.1)\\
\text{subject to} \quad & \sum_{h\in H} y_h \le H_{\max} && && (6.2)\\
& \sum_{v\in V} y_v \le V_{\max} && && (6.3)\\
& y_h \in \{0,1\} && \forall h \in H && (6.4)\\
& y_v \in \{0,1\} && \forall v \in V && (6.5)
\end{aligned}$$

where A represents a set of contamination incidents, $d_{a,\max}$ defines the maximum impact of the contamination incident a, H represents the set of potential hydrant locations, y_h is a binary variable which is 1 if node h is selected as a flushing location, $H_{\max}$ is the maximum number of hydrant locations, V represents the set of potential valve locations, y_v is a binary variable which is 1 if node v is selected as a valving location and $V_{\max}$ is the maximum number of valve locations. The maximum impact of a contamination incident, $d_{a,\max}$, is the total impact across the entire network at the end of the simulation assuming that the contaminant was not detected by a sensor, and no interventions to reduce impacts were implemented. This value is found in the -1 entry of the impact file.

For this problem, hydrants are assumed to be located at any user-defined nodes in the network. In addition, valves are assumed to be located on any user-defined pipes in the network.

6.2 Flushing Solvers

The flushing problem is solved through an iterative optimization process which selects different sets of hydrant and valve locations and evaluates their effectiveness in minimizing the impact of a set of contamination incidents. Two optimization methods, an evolutionary algorithm and a network solver, are available in WST to solve this problem. Each solver is explained in more detail in the following subsections.

6.2.1 Evolutionary Algorithm

The evolutionary algorithm (EA) included with DAKOTA, Coliny EA, is used in the optimization routine for the `flushing` subcommand. Additional information on DAKOTA/Coliny solvers can be found at `http://dakota.sandia.gov/docs/dakota/5.2/html-ref/index.html` and in the DAKOTA user manual (Adams et al., 2013). The random number generator used in Coliny EA is platform dependent. This can result in slight variations in the solution.

To design an EA, the parameter space for the optimization problem is first encoded into a string of numbers. This encoded representation of the problem is called a genetic string, where each element of the genetic string represents one parameter. When the EA is used with the `flushing` subcommand, the parameter space is defined by the number of flushing and valve closure locations. Each location is assigned a sequential integer that represents a feasible location within the EPANET network. The final EA solution is reported based on the EPANET node/pipe IDs.

The EA has several solver options that define how the EA evolves. These options can be set in the `[solver][options]` sections of the `flushing` WST configuration file and are specific to the Coliny EA solver. The EA evolves an initial genetic strings of size `[population_size]` that is set based on `[initialization_type]` using the following steps:

1. Evaluation: Evaluate the solution for each genetic string. This involves function calls to the `tevasim` and `sim2Impact` subcommands for each string to define the impact value.
2. Breeding: Select two members of the population based on fitness. The probability of selection is based on `[fitness_type]`.
3. Crossover: Crossover two members based on `[crossover_type]` and `[crossover_rate]`.
4. Mutation: Mutate the two members based on `[mutation_type]` and `[mutation_rate]`.
5. Steps 2-4 are repeated until the entire population has been changed.
6. Replacement: After a new population is created, the old population is replaced by the current population while keeping the highest ranked string (elitist = 1 replacement option).

Steps 1-6 are repeated until `[max_iterations]` or `[max_function_evaluations]` criteria is met.

6.2.2 Network Solver

The network solver used in WST is a network-constrained, derivative-free local search optimization algorithm. It is a discrete analog-to-pattern search. The allowable moves are to adjacent nodes (or pipes), rather than moves in the continuous space. This approach provides local refinement of candidate solutions. The valid moves include removing a node (or pipe) location and replacing it with one anywhere in the network. Two forms of the network solver can be used: with and without initial starting points. The initial starting points are node (or pipe) locations in the network in which the algorithm should begin its local search. If these points are not supplied to the algorithm, then it reduces to a greedy placement algorithm. Convergence is met when no remaining moves improve the solution.

6.2.3 Flushing Optimization for Large Problems

The iterative optimization process used for flushing requires numerous simulations of the network hydraulics and water quality. Computational run time depends on several factors, including the size of the network model, the number of possible contamination incidents, the number of feasible flushing locations and the solver options. For the network solver,the computational run time also depends on the network model geometry surrounding the local search region. For large flushing optimization problems, several techniques can be used to decrease the computational run time. These options include running multiple instances of the underlying simulation in parallel, setting a stop time criteria and skeletonizing the network model.

6.2.3.1 Parallelization

The Dakota coliny_ea and StateMachineLS solvers both include an option to set the number of threads used to perform the simulations. This option is specified in `[solver][threads]`, as shown below. It should be set to the number of threads that are available for the simulation. The default value is 1. If an integer greater than 1 is specified, Dakota will run that many threads. The expected efficiency of parallelization is nearly linear with the number of threads (up to the number of processor cores in the computer).

```
solver:
  type: coliny_ea
  threads: 2
```

6.2.3.2 Stop time criteria

If a solution needs to be identified within a certain amount of time, a simulation stop time criteria can be defined. This option causes the solver to terminate after a specified time, even if the underlying algorithm has not converged to an optimal solution. The stop time criteria is included with both the Dakota coliny_ea and StateMachineLS solvers, and it is specified in `[solver][options][misc_options]`, as shown below. The max_time has to be given in seconds. As the optimization algorithms only check the elapsed time once per major iteration, the actual solver run time will be slightly longer than the specified maximum run time. When the optimization process is cut off prematurely, the best solution obtained so far is reported to the user.

```
solver:
  type: StateMachineLS
  options:
    misc_options: "’max_time=10’"
```

6.2.3.3 Skeletonization

To reduce the size of the network model, possible contamination incidents and the number of feasible flushing locations, the user can skeletonize the problem and evaluate the results on the full network model. WST includes a utility script, spotSkeleton, that can be used to skeletonize network models (See Executable Files Section 14.5). Network models are skeletonized based on a pipe diameter threshold. The executable, spotSkeleton, creates a new EPANET input (inp) file and a related map file. The map file associates the nodes in the skeletonized network model (upscaled nodes) to the nodes in the original network model (downscaled nodes). When working with a skeletonized network model, other aspects of the flushing problem also need to upscaled based on the skeletonization map. These include: the injection location(s) and strength of the contamination incident(s), the population at each node, the sensor placement, the feasible flushing locations and the initial points for the optimization solver. For example, if Nodes 1, 2, 3 and 4 in the original network model are represented by Node 2 in the skeletonized network model, then a sensor placed at Node 4 in the original network model should be placed at Node 2 in the skeletonized network model.

After flushing optimization is run on the skeletonized network model, the solution can then be evaluated or refined using the original network model. To evaluate the current solution, the locations on the skeletonized network should be listed as the only feasible flushing locations in the original network model and the EVALUATE solver option should be used (See Example 6.4.3). To refine the current solution, downscale the locations on the skeletonized network and use those original network nodes as the only feasible flushing locations in a second optimization. In this refinment, the solver type and solver options can be different for the first and second optimization.

6.3 `flushing` Subcommand

The `flushing` subcommand is executed using the following command line:

```
wst flushing <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst flushing --help
```

6.3.1 Configuration File

The `flushing` subcommand generates a template configuration file using the following command line:

```
wst flushing --template <configfile>
```

The `flushing` template configuration file is shown in Figure 6.2. Brief descriptions of the options are included in the template after the # sign.

6.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `flushing` subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

scenario

location

A list that describes the injection locations for the contamination scenarios. The options are: (1) ALL, which denotes all nodes (excluding tanks and reservoirs) as contamination injection locations; (2) NZD, which denotes all nodes with non-zero demands as contamination injection locations; or (3) an EPANET node ID, which identifies a node as the contamination injection location. This allows for an easy specification of single or multiple contamination scenarios.

Required input unless a TSG or TSI file is specified.

type

The injection type for the contamination scenarios. The options are MASS, CONCEN, FLOWPACED or SETPOINT. See the EPANET 2.00.12 user manual for additional information about source types (Rossman, 2000).

Required input unless a TSG or TSI file is specified.

strength

The amount of contaminant injected into the network for the contamination scenarios. If the type option is MASS, then the units for the strength are in mg/min. If the type option is CONCEN,

```
# flushing configuration template
network:
  epanet file: Net3.inp                  # EPANET 2.00.12 network file name
scenario:
  location: [NZD]                        # Injection location: ALL, NZD or EPANET ID
  type: MASS                             # Injection type: MASS, CONCEN, FLOWPACED or SETPOINT
  strength: 100.0                        # Injection strength [mg/min or mg/L depending on
                                         #   type]
  species: null                          # Injection species, required for EPANET-MSX
  start time: 0                          # Injection start time [min]
  end time: 1440                         # Injection end time [min]
  tsg file: null                         # TSG file name, overrides injection parameters above
  tsi file: null                         # TSI file name, overrides TSG file
  signals: null                          # Signal files, overrides TSG or TSI files
  msx file: null                         # Multi-species extension file name
  msx species: null                      # MSX species to save
  merlion: false                         # Use Merlion as WQ simulator, true or false
impact:
  erd file: null                         # ERD database file name
  metric: [PE]                           # Impact metric
  tai file: Net3_bio.tai                 # Health impact file name, required for public health
                                         #   metrics
  response time: 0                       # Time [min] needed to respond
  detection limit: [0.0]                 # Thresholds needed to perform detection
  detection confidence: 1                # Number of sensors for detection
flushing:
  detection: [111, 127, 179]             # Sensor locations to detect contamination scenarios
  flush nodes:
    feasible nodes: NZD                  # Feasible flushing nodes
    infeasible nodes: NONE               # Infeasible flushing nodes
    max nodes: 2                         # Maximum number of nodes to flush
    rate: 800.0                          # Flushing rate [gallons/min]
    response time: 0.0                   # Time [min] between detection and flushing
    duration: 480.0                      # Flushing duration [min]
  close valves:
    feasible pipes: ALL                  # Feasible pipes to close
    infeasible pipes: NONE               # Infeasible pipes to close
    max pipes: 0                         # Maximum number of pipes to close
    response time: 0.0                   # Time [min] between detection and closing pipes
solver:
  type: StateMachineLS                   # Solver type
  options:                               # A dictionary of solver options
  threads: 1                             # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 6.2: The `flushing` configuration template file.

FLOWPACED or SETPOINT, then units are in mg/L.

Required input unless a TSG or TSI file is specified.

species

The name of the contaminant species injected into the network. This is the name of a single species. It is required when using EPANET-MSX, since multiple species might be simulated, but only one is injected into the network. For cases where multiple contaminants are injected, a TSI file must be used.

Required input for EPANET-MSX unless a TSG or TSI file is specified.

start time

The injection start time that defines when the contaminant injection begins. The time is given in minutes and is measured from the start of the simulation. For example, a value of 60 represents an injection that starts at hour 1 of the simulation.

Required input unless a TSG or TSI file is specified.

end time

The injection end time that defines when the contaminant injection stops. The time is given in minutes and is measured from the start of the simulation. For example, a value of 120 represents an injection that ends at hour 2 of the simulation.

Required input unless a TSG or TSI file is specified.

tsg file

The name of the TSG scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSG file will override the location, type, strength, species, start and end times options specified in the WST configuration file. The TSG file format is documented in File Formats Section 13.12.

Optional input.

tsi file

The name of the TSI scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSI file will override the TSG file, as well as the location, type, strength, species, start and end time options specified in the WST configuration file. The TSI file format is documented in File Formats Section 13.13.

Optional input.

signals

Name of file or directory with information to generate or load signals. If a file is provided the list of inp tsg tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Optional input.

msx file

The name of the EPANET-MSX multi-species file that defines the multi-species reactions to be simulated using EPANET-MSX.

Required input for EPANET-MSX.

msx species

The name of the MSX species whose concentration profile will be saved by the EPANET-MSX simulation and used for later calculations.

Required input for EPANET-MSX.

merlion

A flag to indicate if the Merlion water quality simulator should be used. The options are true or false. If an MSX file is provided, EPANET-MSX will be used.

Required input, default = false.

impact

erd file

The name of the ERD database file that contains the contaminant transport simulation results. It is created by running the `tevasim` subcommand. Multiple ERD files (entered as a list, i.e., [<file1>, <file2>]) can be combined to generate a single impact file. This can be used to combine simulation results from different types of contaminants, in which the ERD files were generated from different TSG files.

Required input.

metric

The impact metric used to compute the impact file. Options include EC, MC, NFD, PD, PE, PK, TD or VC. One impact file is created for each metric selected. These metrics are defined in Section 4.1.

Required input.

tai file

The name of the TAI file that contains health impact information. The TAI file format is documented in File Formats Section 13.11.

Required input if a public health metric is used (PD, PE or PK).

response time

The number of minutes that are needed to respond to the detection of a contaminant. This represents the time that it takes a water utility to stop the spread of the contaminant in the network and eliminate the consumption of contaminated water. As the response time increases, the impact increases because the contaminant affects the network for a greater length of time.

Required input, default = 0 minutes.

detection limit

The minimum concentration that must be exceeded before a sensor can detect a contaminant. There must be one threshold for each ERD file. The units of these detection limits depend on the units of the contaminant simulated for each ERD file (e.g., number of cells of a biological agent).

Required input, default = 0.

detection confidence

The number of sensors that must detect an incident before the impacts are calculated.

Required input, default = 1 sensor.

flushing

detection

The sensor network design used to detect contamination scenarios. The sensor locations are used to compute a detection time for each contamination scenario. The options are a list of EPANET node IDs or a file name which contains a list of EPANET node IDs.

Required input.

flush nodes

feasible nodes

A list that defines the nodes in the network that can be flushed. The options are: (1) ALL, which specifies all nodes as feasible flushing locations; (2) NZD, which specifies all non-zero demand nodes as feasible flushing locations; (3) NONE, which specifies no nodes as feasible flushing locations; (4) a list of EPANET node IDs, which identifies specific nodes as feasible flushing locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as feasible flushing locations.

Required input, default = ALL.

infeasible nodes

A list that defines the nodes in the network that cannot be flushed. The options are: (1) ALL, which specifies all nodes as infeasible flushing locations; (2) NZD, which specifies all non-zero demand nodes as infeasible flushing locations; (3) NONE, which specifies no nodes as infeasible flushing locations; (4) a list of EPANET node IDs, which identifies specific nodes as infeasible flushing locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as infeasible flushing locations.

Optional input, default = NONE.

max nodes

The maximum number of node locations that can be flushed simultaneously in the network. The value is a nonnegative integer or a list of nonnegative integers. When a list is specified, the optimization will be performed for each number in this list. For example, a value of 3 means that a maximum of 3 node will be identified as flushing locations during the optimization process.

Required input.

rate

The flushing rate for each node location to be flushed. A new demand pattern will be created using this rate for the node. The value is a nonnegative integer. For example, a value of 800 means that an additional demand of 800 gpm is applied at a particular node. This rate is applied to all flushing locations identified in the optimization process.

Required input.

response time

The time in minutes between the detection of a contamination incident and the start of flushing. The value is a nonnegative integer. For example, a value of 120 represents a 120 minutes or a 2 hour delay between the time of detection and the start of flushing.

Required input.

duration

The length of time in minutes that flushing will be simulated at a particular node. The value is a nonnegative integer. For example, a value of 240 means that flushing would be simulated at a particular node for 4 hours. This duration is applied to all flushing locations identified in the optimization process.

Required input.

close valves

feasible pipes

A list that defines the pipes in the network that can be closed. The options are: (1) ALL, which specifies all pipes as feasible pipes to close; (2) DIAM min max, which specifies all pipes with a specific diameter as feasible pipes to close; (3) NONE, which specifies no pipes as feasible pipes to close; (4) a list of EPANET pipe IDs, which identifies specific pipes as feasible pipes to close; or (5) a filename, which references a space or comma separated file containing a list of specific pipes as feasible pipes to close.

Required input, default = ALL.

infeasible pipes

A list that defines the pipes in the network that cannot be closed. The options are: (1) ALL, which specifies all pipes as infeasible pipes to close; (2) DIAM min max, which specifies all pipes with a specific diameter as infeasible pipes to close; (3) NONE, which specifies no pipes as infeasible pipes to close; (4) a list of EPANET pipe IDs, which identifies specific pipes as infeasible pipes to close; or (5) a filename, which references a space or comma separated file containing a list of specific pipes as infeasible pipes to close.

Optional input, default = NONE.

max pipes

The maximum number of pipes that can be closed simultaneously in the network. The value must be a nonnegative integer or a list of nonnegative integers. When a list is specified, the optimization will be performed for each number in this list. For example, a value of 2 means that a maximum of 2 pipes to close will be identified during the optimization process.

Required input.

response time

The time in minutes between the detection of a contamination incident and closing pipes. The value is a nonnegative integer. For example, a value of 120 would represent a 120 minutes or a 2 hour delay between the time of detection and the start of pipe closures.

Required input.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

In addition to these standard WST configuration options, the solver block can define an evaluation option. To evaluate the flushing response without solving the optimization problem, the solver type can be set as EVALUATE. This option allows a set of flushing locations to be evaluated against a different contamination scenario than the one for which it was designed.

The solver block can also define specific options for the optimization solver. The solver options should be modified according to the specific optimization problem. If the options are not set in the solver block, then the default values for these options are used. The two solvers available in the `flushing` subcommand each have their own options. The EA solver has numerous options which can be defined. Additional information on the options available for the EA solver can found in the DAKOTA user manual (Adams et al., 2013). An example of the EA solver options are listed below.

```
solver:
  type: coliny_ea
  options:
    crossover_rate: 0.8
    crossover_type: uniform
    fitness_type: linear_rank
    initialization_type: unique_random
    max_function_evaluations: 30000
    max_iterations: 1000
    mutation_rate: 1
    mutation_type: offset_uniform
    population_size: 50
    seed: 11011011
```

The network solver has two options that can be set in the solver block of the configuration file.

```
solver:
  type: StateMachineLS
  options:
    verbosity: 2
    max_fcn_evaluations: 0
```

6.3.3 Subcommand Output

The `flushing` subcommand creates a YAML file called <output prefix>flushing_output.yml that contains an optimized set of node locations (EPANET node IDs) to flush, an optimized set of pipe locations (EPANET pipe IDs) to close, the final impact metric, the run date and CPU time. The log file called <output prefix>flushing_output.log contains basic debugging information. A visualization YAML configuration file named <output prefix>flushing_output_vis.yml is also created. The `visualization` subcommand is automatically run using this YAML file.

6.4 Flushing Response Examples

To demonstrate the two different solvers available in the `flushing` subcommand, two examples are presented. Both examples have the same characteristics in terms of the contamination scenario and flushing parameters. EPANET Example Network 3 (Net3.inp) is the network used and the contamination scenario is an hour long injection at node 101 beginning at hour 3 in the simulation. A maximum of three hydrants can be flushed for a duration of eight hours at a rate 800 gal/min. The option to close pipes/valves was not included in these analyses. The impact metric being minimized is population exposed (PE). In addition, the third and forth examples are provided to demonstrate the evaluate and stop time criteria options, respectively.

6.4.1 Example 1

The first example uses the EA solver (coliny_ea) with the parallization option enabled and the configuration file, flushing_ex1.yml, is shown in Figure 6.3. This example has the `[solver][threads]` option set to 2 threads. Please note: if the computer used to execute the example only has one thread, change the `[solver][threads]` option to 1 instead of 2.

The example can be executed using the following command line:

```
wst flushing flushing_ex1.yml
```

The YAML output file, Net3flushing_output.yml, for example 1 is shown in Figure 6.4. The EA selected to flush nodes 113, 191 and 197 for a PE impact value of 5292. The CPU time was approximately 6 minutes using 2 threads. Since the random number generator used in the EA solver is platform dependent, the solution can be slightly different if the example is executed on a computer with Windows.

6.4.2 Example 2

The second example uses the network solver (StateMachineLS) without initial points and the configuration file, flushing_ex2.yml, is shown in Figure 6.5.

The example can be executed using the following command line:

```
wst flushing flushing_ex2.yml
```

The YAML output file, Net3flushing_output.yml, for example 2 is shown in Figure 6.6. The network solver selected to flush nodes 101, 103 and 109 for a PE impact metric of 4919. The CPU time was approximately 2 minutes.

Examining the output files from the two examples shows that the optimization solvers identified different solutions. As EAs are not guaranteed to find the optimal solution, these results are not unexpected. In addition, the CPU times to obtain the solutions are different. The EA solution took about 6 minutes to obtained using 2 threads, while the network solver solution was achieved in approximately 2 minutes.

6.4.3 Example 3

The third example uses the evaluate option and the configuration file, flushing_ex3.yml, is shown in Figure 6.7. In this example, the same contamination scenario is used but only two (2) flushing locations are evaluated in terms of reducing the PE impact metric. The flushing locations being evaluated are nodes 101 and 127.

The example can be executed using the following command line:

```
wst flushing flushing_ex3.yml
```

The YAML output file, Net3flushing_output.yml, for example 3 is shown in Figure 6.8. These two flushing locations resulted in a PE impact metric of 10,759. Compared to the results from example 2, the PE impact metric is much larger since only two flushing locations are used instead of three. The evaluate option can be used to compare the impact metrics obtained from various flushing location combinations.

6.4.4 Example 4

The fourth example demonstrates the stop time option and uses almost the same configuration file as example 2. The configuration file, flushing_ex4.yml, is shown in Figure 6.9, in which the stop criteria option is enabled using the `[solver]` `[options]` `[misc_options]` option set to "”max_time=30”".

The example can be executed using the following command line:

```
wst flushing flushing_ex4.yml
```

The YAML output file, Net3flushing_output.yml, for example 4 is shown in Figure 6.10. The network solver selected to flush node 105 for a PE impact value of 9676. The CPU time was 45 seconds, which is greater than the stop time criteria since it is only checked periodically. The flushing solution was different than the solution obtained from example 2, since the stop time option was not used and it executed the optimization process to completion.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: [’101’]
  type: MASS
  strength: 1.450000e+010
  species: null
  start time: 180
  end time: 240
  tsg file: null
  tsi file: null
  msx file: null
  msx species: null
  merlion: false
impact:
  erd file: null
  metric: [PE]
  tai file: Net3/Net3_bio.tai
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: null
flushing:
  detection: [’111’, ’127’, ’179’]
  flush nodes:
    feasible nodes: NZD
    infeasible nodes: NONE
    max nodes: 3
    rate: 800.0
    response time: 0.0
    duration: 480.0
  close valves:
    feasible pipes: NONE
    infeasible pipes: NONE
    max pipes: 0
    response time: 0.0
solver:
  type: coliny_ea
  threads: 2
  options:
    crossover_rate: 0.8
    crossover_type: uniform
    fitness_type: linear_rank
    initialization_type: unique_random
    max_function_evaluations: 1000
    max_iterations: 1000
    mutation_rate: 1
    mutation_type: offset_uniform
    population_size: 50
    seed: 11011011
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/flushing_ex1/Net3
  debug: 0
```

Figure 6.3: The `flushing` configuration file for example 1.

```
# flushing output
general:
  version: 1.5                       # WST version
  date: ’2019-03-05’                 # Run date
  cpu time: 345.938                  # CPU time (sec)
  directory: C:/wst-1.5/examples/flushing_ex1
  log file: Net3flushing_output.log  # Log file
flushing:
  nodes: [’197’, ’191’, ’113’]       # List of nodes to flush
  pipes: []                          # List of pipes to close
  objective: 5292.66                 # Objective value
```

Figure 6.4: The `flushing` YAML output file for example 1.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: [’101’]
  type: MASS
  strength: 1.450000e+010
  species: null
  start time: 180
  end time: 240
  tsg file: null
  tsi file: null
  msx file: null
  msx species: null
  merlion: false
impact:
  erd file: null
  metric: [PE]
  tai file: Net3/Net3_bio.tai
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: null
flushing:
  detection: [’111’, ’127’, ’179’]
  flush nodes:
    feasible nodes: NZD
    infeasible nodes: NONE
    max nodes: 3
    rate: 800.0
    response time: 0.0
    duration: 480.0
  close valves:
    feasible pipes: NONE
    infeasible pipes: NONE
    max pipes: 0
    response time: 0.0
solver:
  type: StateMachineLS
  threads: 1
  options:
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/flushing_ex2/Net3
  debug: 0
```

Figure 6.5: The `flushing` configuration file for example 2.

```
# flushing output
general:
  version: 1.5                       # WST version
  date: ’2019-03-05’                 # Run date
  cpu time: 119.257                  # CPU time (sec)
  directory: C:/wst-1.5/examples/flushing_ex2
  log file: Net3flushing_output.log  # Log file
flushing:
  nodes: [’101’, ’103’, ’109’]       # List of nodes to flush
  pipes: []                          # List of pipes to close
  objective: 4918.76                 # Objective value
```

Figure 6.6: The `flushing` YAML output file for example 2.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: [’101’]
  type: MASS
  strength: 1.450000e+010
  species: null
  start time: 180
  end time: 240
  tsg file: null
  tsi file: null
  msx file: null
  msx species: null
  merlion: false
impact:
  erd file: null
  metric: [PE]
  tai file: Net3/Net3_bio.tai
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: null
flushing:
  detection: [’111’, ’127’, ’179’]
  flush nodes:
    feasible nodes: [’101’, ’127’]
    infeasible nodes: NONE
    max nodes: 2
    rate: 800.0
    response time: 0.0
    duration: 480.0
  close valves:
    feasible pipes: NONE
    infeasible pipes: NONE
    max pipes: 0
    response time: 0.0
solver:
  type: EVALUATE
  options:
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/flushing_ex3/Net3
  debug: 0
```

Figure 6.7: The `flushing` configuration file for example 3.

```
# flushing output
general:
  version: 1.5                       # WST version
  date: ’2019-03-04’                 # Run date
  cpu time: 3.715                    # CPU time (sec)
  directory: C:/wst-1.5/examples/flushing_ex3
  log file: Net3flushing_output.log  # Log file
flushing:
  nodes: [’101’, ’127’]              # List of nodes to flush
  pipes: []                          # List of pipes to close
  objective: 10758.9                 # Objective value
```

Figure 6.8: The `flushing` YAML output file for example 3.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: [’101’]
  type: MASS
  strength: 1.450000e+010
  species: null
  start time: 180
  end time: 240
  tsg file: null
  tsi file: null
  msx file: null
  msx species: null
  merlion: false
impact:
  erd file: null
  metric: [PE]
  tai file: Net3/Net3_bio.tai
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: null
flushing:
  detection: [’111’, ’127’, ’179’]
  flush nodes:
    feasible nodes: NZD
    infeasible nodes: NONE
    max nodes: 3
    rate: 800.0
    response time: 0.0
    duration: 480.0
  close valves:
    feasible pipes: NONE
    infeasible pipes: NONE
    max pipes: 0
    response time: 0.0
solver:
  type: StateMachineLS
  threads: 1
  options:
    misc_options: "’max_time=30’"
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/flushing_ex4/Net3
  debug: 0
```

Figure 6.9: The `flushing` configuration file for example 4.

```
# flushing output
general:
  version: 1.5                       # WST version
  date: ’2019-03-05’                 # Run date
  cpu time: 40.109                   # CPU time (sec)
  directory: C:/wst-1.5/examples/flushing_ex4
  log file: Net3flushing_output.log  # Log file
flushing:
  nodes: [’105’]                     # List of nodes to flush
  pipes: []                          # List of pipes to close
  objective: 9676.3                  # Objective value
```

Figure 6.10: The `flushing` YAML output file for example 4.

Chapter 7

Booster Station Placement

Disinfection booster stations are commonly used throughout water distribution networks to maintain drinking water standards. Disinfectants degrade as they move through the system and booster stations, designed to inject disinfectant at strategic locations, help maintain residual levels. Booster stations can also be placed with water security objectives in mind. WST includes two booster subcommands, `booster_msx` and `booster_mip` that are designed to place booster stations to minimize the impact of a contamination incident. These subcommands use different approaches to model the reaction dynamics between a contaminant and disinfectant.

The `booster_msx` subcommand uses EPANET-MSX to simulate the reaction dynamics between the contaminant and disinfectant. A flowchart representation of the `booster_msx` subcommand is shown in Figure 7.1. The `booster_msx` subcommand employs an iterative process that combines contaminant transport, impact assessment and optimization. The optimization process identifies a set of booster station locations where disinfectant is injected. The contaminant and disinfectant injections and their reaction dynamics are simulated in the contaminant transport process and the effectiveness of the booster injections are evaluated based upon the impact assessment process. Since the `booster_msx` subcommand relies on the `tevasim` and `sim2Impact` subcommands, their required input is also required for the `booster_msx` subcommand. Additionally, the sensor network design used to detect the contamination incident(s) and the booster station characteristics are required inputs. The utility network model is defined by a EPANET 2.00.12 INP file, while the rest of the input can be specified in the `booster_msx` WST configuration file.

The `booster_mip` subcommand uses the linear water quality model Merlion and assumes the reaction dynamics between the contaminant and disinfectant can be approximated by a neutralization or limiting reagent reaction model. A flowchart representation of the `booster_mip` subcommand is shown in Figure 7.2. The utility network model is defined by a EPANET 2.00.12 INP file. Additional input specified in the `booster_mip` WST configuration file are the contamination scenarios, the sensor network design used to detect the contamination incident(s) and the booster station characteristics.

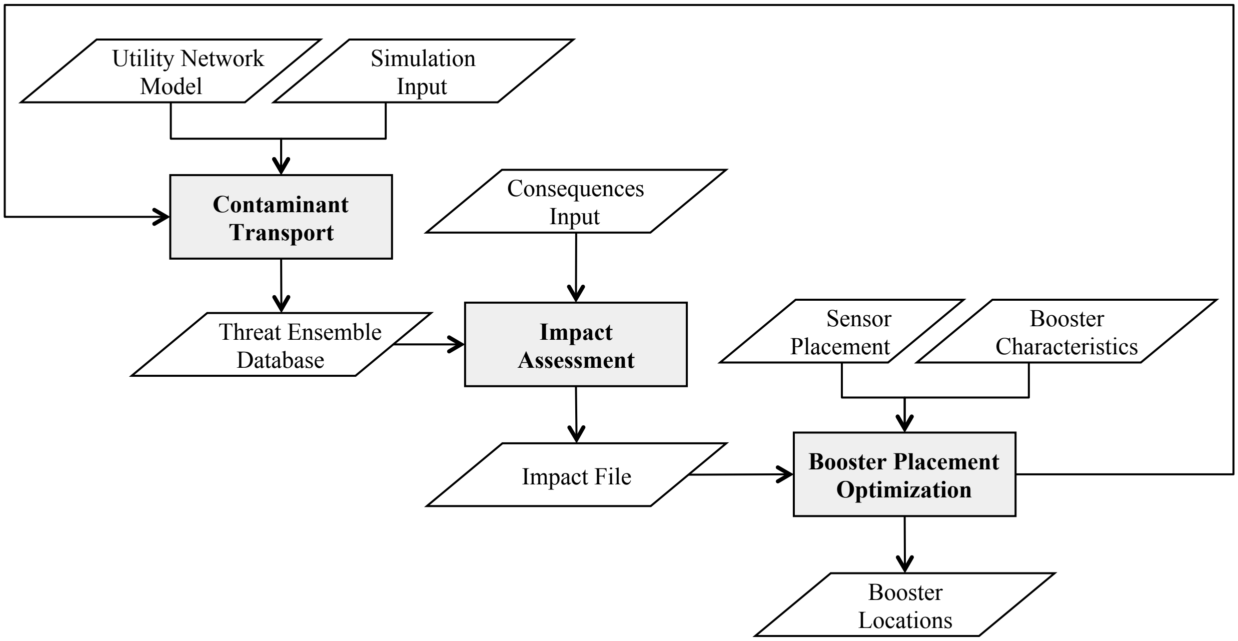


Figure 7.1: Multi-species reaction booster placement flowchart.

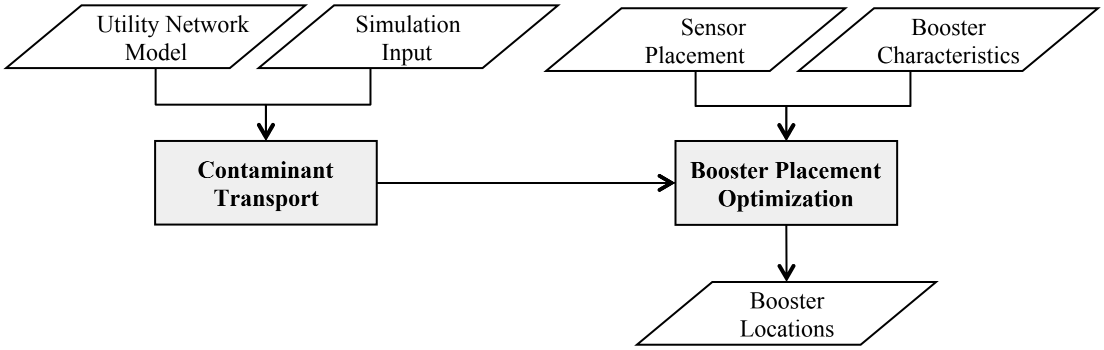


Figure 7.2: MIP booster placement flowchart.

7.1 Booster Placement Using Multi-species Reaction

The `booster_msx` subcommand uses optimization methods integrated with EPANET-MSX to evaluate booster placements using multi-species reaction dynamics between the contaminant and disinfectant. The booster station placement problem formulation selects a set of booster station locations that minimizes the average impact of all contamination incidents given a set of potential booster stations that inject a disinfecting agent. The mathematical formulation can be written as follows:

$$\text{minimize} \quad \frac{1}{A}\sum_{a\in A} d_{a,\max} \tag{7.1}$$

$$\text{subject to} \quad \sum_{b\in B} y_b \leq B_{\max} \tag{7.2}$$

$$y_b \in \{0,1\} \qquad \forall b \in B \tag{7.3}$$

where A represents a set of contamination incidents, $d_{a,\max}$ defines the maximum impact of the contamination incident a, B represents the set of potential booster station locations, y_b is a binary variable which is 1 if node b is selected as a booster station location and $B_{\max}$ is the maximum number of booster station locations. The maximum impact of a contamination incident, $d_{a,\max}$, is the total impact across the entire network at the end of the simulation assuming that the contaminant was not detected by a sensor, and no interventions to reduce impacts were implemented. This value is found in the -1 entry of the impact file. For this problem, it is assumed that booster stations can be located at any user-defined nodes in the network.

7.1.1 Booster MSX Solvers

The multi-species booster station placement problem is solved through an iterative optimization process which selects different sets of booster station locations and evaluates their effectiveness in minimizing the impact of a set of contamination incidents. Two optimization methods, an evolutionary algorithm and a network solver, are available in WST to solve this problem. Each solver is explained in more detail in the following subsections.

7.1.1.1 Evolutionary Algorithm

The evolutionary algorithm (EA) included with DAKOTA, Coliny EA, is used in the optimization routine for the `booster_msx` subcommand. Additional information on DAKOTA/Coliny solvers can be found at `http://dakota.sandia.gov/docs/dakota/5.2/html-ref/index.html` and in the DAKOTA user manual (Adams et al., 2013). The random number generator used in Coliny EA is platform dependent. This can result in slight variations in the solution.

To design an EA, the parameter space for the optimization problem is first encoded into a string of numbers. This encoded representation of the problem is called a genetic string, where each element of the genetic string represents one parameter. When the EA is used with `booster_msx` the parameter space is defined by the number of booster station locations. Each location is assigned a sequential integer that represents a feasible location within the EPANET network. The final EA solution is translated to represent EPANET node IDs.

The EA has several solver options that define how the EA evolves. These options can be set in the `[solver][options]` sections of the `booster_msx` WST configuration file and are specific to the Coliny EA solver. The EA evolves an initial genetic strings of size `[population_size]` that is set based on `[initialization_type]` using the following steps:

1. Evaluation: Evaluate the solution for each genetic string. This involves function calls to the `tevasim` and `sim2Impact` subcommands for each string to define the fitness score.
2. Breeding: Select two members of the population based on fitness. The probability of selection is based on `[fitness_type]`.
3. Crossover: Crossover two members based on `[crossover_type]` and `[crossover_rate]`.
4. Mutation: Mutate the two members based on `[mutation_type]` and `[mutation_rate]`.
5. Steps 2-4 are repeated until the entire population has been changed.
6. Replacement: After a new population is created, the old population is replaced by the current population while keeping the highest ranked string (elitist = 1 replacement option).

Steps 1-6 are repeated until `[max_iterations]` or `[max_function_evaluations]` criteria is met.

7.1.1.2 Network Solver

The network solver used in WST is a network-constrained, derivative-free local search optimization algorithm. It is a discrete analog-to-pattern search. The allowable moves are to adjacent nodes, rather than moves in the continuous space. This approach provides local refinement of candidate solutions. The valid moves include removing a node location and replacing it with one anywhere in the network. Two forms of the network solver can be used: with and without initial starting points. The initial starting points are node locations in the network in which the algorithm should begin its local search. If these points are not supplied to the algorithm, then it reduces to a greedy placement algorithm. Convergence is met when no remaining moves improve the solution.

7.1.2 `booster_msx` Subcommand

The `booster_msx` subcommand is executed using the following command line:

```
wst booster_msx <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst booster_msx --help
```

7.1.2.1 Configuration File

The `booster_msx` subcommand generates a template configuration file using the following command line:

```
wst booster_msx --template <configfile>
```

The `booster_msx` template configuration file is shown in Figure 7.3. Brief descriptions of the options are included in the template after the # sign.

7.1.2.2 Configuration Options

Full descriptions of the WST configuration options used by the `booster_msx` subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

scenario

location

A list that describes the injection locations for the contamination scenarios. The options are: (1) ALL, which denotes all nodes (excluding tanks and reservoirs) as contamination injection locations; (2) NZD, which denotes all nodes with non-zero demands as contamination injection locations; or (3) an EPANET node ID, which identifies a node as the contamination injection location. This allows for an easy specification of single or multiple contamination scenarios.

Required input unless a TSG or TSI file is specified.

type

The injection type for the contamination scenarios. The options are MASS, CONCEN, FLOWPACED or SETPOINT. See the EPANET 2.00.12 user manual for additional information about source types (Rossman, 2000).

```
# booster_msx configuration template
network:
  epanet file: Net3.inp                 # EPANET 2.00.12 network file name
scenario:
  location: [’101’]                     # Injection location: ALL, NZD or EPANET ID
  type: MASS                            # Injection type: MASS, CONCEN, FLOWPACED or SETPOINT
  strength: 100.0                       # Injection strength [mg/min or mg/L depending on
                                        #   type]
  species: BIO                          # Injection species, required for EPANET-MSX
  start time: 0                         # Injection start time [min]
  end time: 1440                        # Injection end time [min]
  tsg file: null                        # TSG file name, overrides injection parameters above
  tsi file: null                        # TSI file name, overrides TSG file
  signals: null                         # Signal files, overrides TSG or TSI files
  msx file: Net3_bio.msx                # Multi-species extension file name
  msx species: BIO                      # MSX species to save
  merlion: false                        # Use Merlion as WQ simulator, true or false
impact:
  erd file: null                        # ERD database file name
  metric: [MC]                          # Impact metric
  tai file: null                        # Health impact file name, required for public health
                                        #   metrics
  response time: 0                      # Time [min] needed to respond
  detection limit: [0.0]                # Thresholds needed to perform detection
  detection confidence: 1               # Number of sensors for detection
booster msx:
  detection: [111, 127, 179]            # Sensor locations to detect contamination scenarios
  toxin species: BIO                    # Toxin species injected in each contaminant scenario
  decon species: CLF                    # Decontaminant injected from booster station
  feasible nodes: ALL                   # Feasible booster nodes
  infeasible nodes: NONE                # Infeasible booster nodes
  max boosters: 2                       # Maximum number of booster stations
  type: FLOWPACED                       # Booster source type: FLOWPACED
  strength: 4.0                         # Booster source strength [mg/L]
  response time: 0.0                    # Time [min] between detection and booster injection
  duration: 600.0                       # Time [min] for booster injection
solver:
  type: coliny_ea                       # Solver type
  options:                              # A dictionary of solver options
  threads: 1                            # Number of concurrent threads or function evaluations
  logfile: null                         # Redirect solver output to a logfile
  verbose: 0                            # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                   # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                              # Debugging level, default = 0
```

Figure 7.3: The `booster_msx` configuration template file.

Required input unless a TSG or TSI file is specified.

strength

The amount of contaminant injected into the network for the contamination scenarios. If the type option is MASS, then the units for the strength are in mg/min. If the type option is CONCEN, FLOWPACED or SETPOINT, then units are in mg/L.

Required input unless a TSG or TSI file is specified.

species

The name of the contaminant species injected into the network. This is the name of a single species. It is required when using EPANET-MSX, since multiple species might be simulated, but only one is injected into the network. For cases where multiple contaminants are injected, a TSI file must be used.

Required input for EPANET-MSX unless a TSG or TSI file is specified.

start time

The injection start time that defines when the contaminant injection begins. The time is given in minutes and is measured from the start of the simulation. For example, a value of 60 represents an injection that starts at hour 1 of the simulation.

Required input unless a TSG or TSI file is specified.

end time

The injection end time that defines when the contaminant injection stops. The time is given in minutes and is measured from the start of the simulation. For example, a value of 120 represents an injection that ends at hour 2 of the simulation.

Required input unless a TSG or TSI file is specified.

tsg file

The name of the TSG scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSG file will override the location, type, strength, species, start and end times options specified in the WST configuration file. The TSG file format is documented in File Formats Section 13.12.

Optional input.

tsi file

The name of the TSI scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSI file will override the TSG file, as well as the location, type, strength, species, start and end time options specified in the WST configuration file. The TSI file format is documented in File Formats Section 13.13.

Optional input.

signals

Name of file or directory with information to generate or load signals. If a file is provided the list of inp tsg tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Optional input.

msx file

The name of the EPANET-MSX multi-species file that defines the multi-species reactions to be simulated using EPANET-MSX.

Required input for EPANET-MSX.

msx species

The name of the MSX species whose concentration profile will be saved by the EPANET-MSX simulation and used for later calculations.

Required input for EPANET-MSX.

merlion

A flag to indicate if the Merlion water quality simulator should be used. The options are true or false. If an MSX file is provided, EPANET-MSX will be used.

Required input, default = false.

impact

erd file

The name of the ERD database file that contains the contaminant transport simulation results. It is created by running the `tevasim` subcommand. Multiple ERD files (entered as a list, i.e., [<file1>, <file2>]) can be combined to generate a single impact file. This can be used to combine simulation results from different types of contaminants, in which the ERD files were generated from different TSG files.

Required input.

metric

The impact metric used to compute the impact file. Options include EC, MC, NFD, PD, PE, PK, TD or VC. One impact file is created for each metric selected. These metrics are defined in Section 4.1.

Required input.

tai file

The name of the TAI file that contains health impact information. The TAI file format is documented in File Formats Section 13.11.

Required input if a public health metric is used (PD, PE or PK).

response time

The number of minutes that are needed to respond to the detection of a contaminant. This represents the time that it takes a water utility to stop the spread of the contaminant in the network and eliminate the consumption of contaminated water. As the response time increases, the impact increases because the contaminant affects the network for a greater length of time.

Required input, default = 0 minutes.

detection limit

The minimum concentration that must be exceeded before a sensor can detect a contaminant. There must be one threshold for each ERD file. The units of these detection limits depend on the units of the contaminant simulated for each ERD file (e.g., number of cells of a biological agent).

Required input, default = 0.

detection confidence

The number of sensors that must detect an incident before the impacts are calculated.

Required input, default = 1 sensor.

booster msx

detection

The sensor network design used to detect contamination scenarios. The sensor locations are used to compute a detection time for each contamination scenario in the TSG file. The options are a list of EPANET node IDs or a file name which contains a list of EPANET node IDs.

Required input.

toxin species

The name of the contaminant species that is injected in each contamination scenario. This is the species that interacts with the injected disinfectant and whose impact is going to be minimized.

Required input.

decon species

The name of the decontaminant or disinfectant species that is injected from the booster stations.

Required input.

feasible nodes

A list that defines nodes that can be considered for the booster station placement problem. The options are: (1) ALL, which specifies all nodes as feasible booster station locations; (2) NZD, which specifies all non-zero demand nodes as feasible booster station locations; (3) NONE, which specifies no nodes as feasible booster station locations; (4) a list of EPANET node IDs, which identifies specific nodes as feasible booster station locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as feasible booster station locations.

Required input, default = ALL.

infeasible nodes

A list that defines nodes that cannot be considered for the booster station placement problem. The options are: (1) ALL, which specifies all nodes as infeasible booster station locations; (2) NZD, which specifies non-zero demand nodes as infeasible booster station locations; (3) NONE, which specifies no nodes as infeasible booster station locations; (4) a list of EPANET node IDs, which identifies specific nodes as infeasible booster station locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as infeasible booster station locations.

Optional input, default = NONE.

max boosters

The maximum number of booster stations that can be placed in the network. The value must be a nonnegative integer or a list of nonnegative integers. When a list is specified, the optimization will be performed for each number in this list.

Required input.

type

The injection type for the disinfectant at the booster stations. The option is FLOWPACED. See the EPANET 2.00.12 user manual for additional information about source types Rossman (2000).

Required input.

strength

The amount of disinfectant injected into the network from the booster stations. For the source type FLOWPACED, the strength units are in mg/L.

Required input.

response time

The time in minutes between the detection of a contamination incident and the start of injecting disinfectants from the booster stations. The value is a nonnegative integer. For example, a value of 120 represents a 120 minutes or a 2 hour delay between the time of detection and the start of booster injections.

Required input.

duration

The length of time in minutes that disinfectant will be injected at the booster stations during the simulation. The value is a nonnegative integer. For example, a value of 240 means that a booster would simulate injection of disinfectant at a particular node for 4 hours. This duration is applied to all booster station locations identified in the optimization process.

Required input.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

In addition to these standard WST configuration options, the solver block can define an evaluation option. To evaluate the placement of the booster stations without solving the optimization problem, the solver type can be set as EVALUATE. This option allows a booster placement to be evaluated against a different contamination incident than which it was designed.

The solver block can also define specific options for the optimization solver. The solver options should be modified according to the specific optimization problem. If the options are not set in the solver block, then the default values for these options are used. The EA solver available in the `booster_msx` subcommand has numerous options which can be defined. Additional information on the options available for the EA solver can found in the DAKOTA user manual (Adams et al., 2013). An example of the EA solver options are listed below.

```
solver:
  type: coliny_ea
  options:
    crossover_rate: 0.8
    crossover_type: uniform
    fitness_type: linear_rank
    initialization_type: unique_random
    max_function_evaluations: 30000
    max_iterations: 1000
    mutation_rate: 1
    mutation_type: offset_uniform
    population_size: 50
    seed: 11011011
```

The network solver has two options that can be set in the solver block of the configuration file.

```
solver:
  type: StateMachineLS
  options:
    verbosity: 2
    max_fcn_evaluations: 0
```

7.1.2.3 Subcommand Output

The `booster_msx` subcommand creates a YAML file called <output prefix>booster_msx_output.yml that contains an optimized list of node locations (EPANET node IDs) to inject the disinfectant, the final impact metric, the run date and CPU time. The log file called <output prefix>booster_msx_output.log contains basic debugging information. A visualization YAML configuration file named <output prefix>booster_msx_output_vis.yml is also created. The `visualization` subcommand is automatically run using this YAML file.

7.2 Booster Placement Using Neutralization or Limiting Reagent Reaction

If either the contaminant or the disinfectant are present in the water distribution system in excess (i.e., there is a clear limiting reagent), the booster station placement problem can be formulated as a mixed-integer program (MIP). The `booster_mip` subcommand uses this MIP to determine optimal node locations of booster stations used for responding to contamination incidents.

Two separate model formulations are available within the `booster_mip` subcommand. These are referred to as the neutralization (NEUTRAL) formulation and the limiting reagent (LIMIT) formulation. Each has a unique set of advantages that can be leveraged depending on the needs of the user. The NEUTRAL formulation models the idealized situation in which the disinfecting agent (e.g., chlorine) immediately inactivates any amount of contaminant it comes into contact with; the amount of disinfectant available for inactivation is not considered. This allows for a highly compact model formulation, which is tractable for application to both large water distribution systems and large scenario ensembles. The placements resulting from the NEUTRAL formulation represent booster station locations that are optimal when the amount of disinfectant required to perform inactivation is small and the time required for complete inactivation is fast.

The LIMIT formulation models the case where the disinfectant is consumed during the reaction with the contaminant. This is more realistic than the NEUTRAL formulation in that the optimal solution is highly dependent on the amount of disinfectant injected by the booster stations. However, the model still assumes that upon mixing the limiting reagent is completed consumed, leaving the excess of the other species. The LIMIT formulation includes a stoichiometric ratio, which represents the mass of disinfectant removed per mass of contaminant removed. The units for disinfectant mass and contaminant mass are determined by the type of injection used for each species (mg for chemical and CFU for biological). The stoichiometric ratio can be adjusted to approximate a more realistic pairing of specific disinfectant and contaminant species (e.g., chlorine and $E.\ coli$).

The optimization is performed over an ensemble of contamination incidents. The `booster_mip` subcommand uses Merlion to perform water quality simulations, which are used to generate the necessary data for the optimization formulation. The amount of time required for simulations can differ depending on the problem formulation selected by the user (e.g., LIMIT or NEUTRAL).

7.2.1 Neutralization NEUTRAL Formulation

The NEUTRAL formulation is as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{a\in A}\alpha_a \sum_{n\in N}\sum_{t\in T}\delta_{n,t,a}m_{n,t,a} && \text{where } m_{n,t,a} = c_{n,t,a}d_{n,t} && (7.4)\\
&\text{subject to} && \delta_{n,t,a} \geq 1 - \sum_{b\in B} y_b Z_{n,t,a,b} && \forall n\in N, t\in T, a\in A && (7.5)\\
& && \sum_{b\in B} y_b \leq B_{\max} && && (7.6)\\
& && 0 \leq \delta_{n,t,a} \leq 1 && \forall n\in N, t\in T, a\in A && (7.7)\\
& && y_b \in \{0,1\} && \forall b\in B && (7.8)
\end{aligned}$$

where A represents the set of scenarios, N defines the set of network nodes, T represents the set of time steps, B defines the set of potential booster station locations, α_a is the probability of scenario a, $m_{n,t,a}$ and $c_{n,t,a}$ are the mass and concentration, respectively, of contaminant leaving node n at time step t for scenario a, $d_{n,t}$ is the demand for the corresponding node and time step and $B_{\max}$ is the maximum number of booster stations. The continuous variable, $\delta_{n,t,a}$, indicates whether the contaminant is available for consumption at node n and time step t for scenario a and the binary variable, y_b, is 1 if node b is selected as a booster station location. In addition, $Z_{n,t,a,b}$ is determined from the pre-computed booster station simulations. These simulations determine the node-time pairs that are neutralized based on the specific contaminant scenario and feasible booster station locations. The parameter $Z_{n,t,a,b}$ is equal to 1 only if a booster station installed at node b neutralizes the contaminant leaving node n and time step t for scenario a, otherwise, $Z_{n,t,a,b}$ is 0.

Equation 7.4 is the objective function, which minimizes the mass consumed across all nodes, for every scenario, for every time step in the simulation. Equation 7.5 ensures that $\delta_{n,t,a}$ equals 0 if at least one selected disinfectant booster station location provides neutralization of node n at time step t for scenario a, otherwise, $\delta_{n,t,a}$ equals 1. Equation 7.6 restricts the number of booster stations to be less than or equal to $B_{\max}$ and Equations 7.7 and 7.8 limit the range for $\delta_{n,t,a}$ and y_b.

Contaminant and disinfectant simulations are computed by the `booster_mip` subcommand using Merlion. These simulations define the parameters $Z_{n,t,a,b}$ and $m_{n,t,a}$. Similar simulations and parameters could be obtained using EPANET 2.00.12. However, the linear water quality model defined in Merlion is necessary for the limiting reagent model.

7.2.2 Limiting Reagent LIMIT Formulation

The LIMIT formulation is as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{a \in A} \alpha_a \sum_{n \in N} \sum_{t \in T} c^{\text{con}}_{n,t,a} d_{n,t} && && (7.9)\\
\text{subject to} \quad & G c^{\text{con}}_{n,t,a} = D(m^{\text{con}}_{n,t,a} - r^{\text{con}}_{n,t,a}) && \forall a \in A && (7.10)\\
& G c^{\text{decon}}_{n,t,a} = D(m^{\text{decon}}_{n,t,a} - \sigma r^{\text{con}}_{n,t,a}) && \forall a \in A && (7.11)\\
& m^{\text{decon}}_{b,t,a} = y_b I_{b,t,a} && \forall b \in B, t \in T, a \in A && (7.12)\\
& m^{\text{decon}}_{n,t,a} = 0 && \forall n \in N \setminus B, t \in T, a \in A && (7.13)\\
& \sum_{b \in B} y_b \le B_{\max} && && (7.14)\\
& c^{\text{con}}_{n,t,a}, c^{\text{decon}}_{n,t,a} \ge 0 && \forall n \in N, t \in T, a \in A && (7.15)\\
& r^{\text{con}}_{n,t,a} \ge 0 && \forall n \in N, t \in T, a \in A && (7.16)\\
& y_b \in \{0, 1\} && \forall b \in B && (7.17)
\end{aligned}
$$

where A represents the set of scenarios, N defines the set of network nodes, T represents the set of time steps, B defines the set of potential booster locations, α_a is the probability of scenario a, $d_{n,t}$ is the demand at node n and time step t, $r_{n,t,a}$ is the mass of contaminant removed at node n and time step t for scenario a (based on the reaction dynamics between the contaminant and disinfectant), σ is the stoichiometric ratio for the reaction dynamics (mass unit of disinfectant removed per mass unit of contaminant removed) and $B_{\max}$ is the maximum number of booster stations. In addition, $c^{\text{con}}_{n,t,a}$ and $c^{\text{decon}}_{n,t,a}$ are the concentrations of contaminant and disinfectant, respectively, at node n and time step t for scenario a. The variables $m^{\text{con}}_{n,t,a}$ and $m^{\text{decon}}_{n,t,a}$ are the mass injection profiles for the contaminant and disinfectant, respectively, at node n and time step t for scenario a. The parameters G and D are matrices that define the water quality model. They form the mathematical relationship between the input mass injected and the output concentration at all nodes and times. These are discussed in more detail in Section 12.1. A first order decay rate can be added to the contaminant and disinfectant. The binary variable, y_b, is 1 if node b is selected as a booster station location. The variable $I_{b,t,a}$ is the injection profile for booster b at time step t for scenario a.

Equation 7.9 is the objective function, which minimizes the mass consumed across all nodes, for every scenario, for every time step in the simulation. Equations 7.10 and 7.11 include the embedded linear water quality model, as stored in the G and D matrices. Equation 7.12 sets the injection profile if a booster is placed. Equation 7.13 sets the injection profile to 0 if a booster is not placed. The variable $N \setminus B$ is the set of nodes that are not potential booster station locations. Equation 7.14 restricts the number of booster stations to be less than or equal to $B_{\max}$. Equation 7.15 places bounds on the contaminant and disinfectant concentrations. Equation 7.16 places bounds on the disinfectant mass injection. Equation 7.17 defines y_b as a binary variable.

7.2.3 Booster MIP Solvers

The `booster_mip` subcommand requires a standard MIP solver to perform booster station placement. A variety of public domain and commercial MIP solvers exist that can be used with the `booster_mip` subcommand, including GLPK, CBC, PICO, CPLEX, GUROBI and XPRESS.

The modeling language, specified by the model format option in the booster mip block of the configuration file, determines the true list of solvers available for booster station placement. The following shows examples of solvers available with AMPL (Fourer et al., 2002) and Pyomo (Hart et al., 2012):

```
Solver   [type]       [model format]
====================================
GLPK     glpk           PYOMO
CPLEX    cplex          PYOMO
CPLEX    cplexamp       PYOMO
CPLEX    cplexamp      AMPL
GUROBI   gurobi        PYOMO
GUROBI   gurobi_ampl   PYOMO
GUROBI   gurobi_ampl   AMPL
CBC      cbc           PYOMO
CBC      cbc           AMPL
```

Documentation for AMPL (Fourer et al., 2002) and Pyomo (Hart et al., 2012) can provide more information about the solvers available with these modeling languages.

7.2.4 `booster_mip` Subcommand

The `booster_mip` subcommand is executed using the following command line:

```
wst booster_mip <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst booster_mip --help
```

7.2.4.1 Configuration File

The `booster_mip` subcommand generates a template configuration file using the following command line:

```
wst booster_mip --template <configfile>
```

The `booster_mip` template configuration file is shown in Figure 7.4. Brief descriptions of the options are included in the template after the # sign.

7.2.4.2 Configuration Options

Full descriptions of the WST configuration options used by the `booster_mip` subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

scenario

location

A list that describes the injection locations for the contamination scenarios. The options are: (1) ALL, which denotes all nodes (excluding tanks and reservoirs) as contamination injection locations; (2) NZD, which denotes all nodes with non-zero demands as contamination injection locations; or (3) an EPANET node ID, which identifies a node as the contamination injection location. This allows for an easy specification of single or multiple contamination scenarios.

Required input unless a TSG or TSI file is specified.

type

The injection type for the contamination scenarios. The options are MASS, CONCEN, FLOWPACED or SETPOINT. See the EPANET 2.00.12 user manual for additional information about source types (Rossman, 2000).

Required input unless a TSG or TSI file is specified.

strength

```
# booster_mip configuration template
network:
  epanet file: Net3.inp                  # EPANET 2.00.12 network file name
scenario:
  location: null                         # Injection location: ALL, NZD or EPANET ID
  type: null                             # Injection type: MASS, CONCEN, FLOWPACED or SETPOINT
  strength: null                         # Injection strength [mg/min or mg/L depending on
                                         #   type]
  species: null                          # Injection species, required for EPANET-MSX
  start time: null                       # Injection start time [min]
  end time: null                         # Injection end time [min]
  tsg file: Net3.tsg                     # TSG file name, overrides injection parameters above
  tsi file: null                         # TSI file name, overrides TSG file
  signals: null                          # Signal files, overrides TSG or TSI files
  msx file: null                         # Multi-species extension file name
  msx species: null                      # MSX species to save
  merlion: false                         # Use Merlion as WQ simulator, true or false
booster mip:
  detection: [111, 127, 179]             # Sensor locations to detect contamination scenarios
  model type: NEUTRAL                    # Booster model type: NEUTRAL or LIMIT
  model format: PYOMO                    # Booster optimization model: AMPL or PYOMO
  stoichiometric ratio: [0.0]            # Stoichiometric ratio [decon/toxin], LIMIT model only
  objective: MC                          # Objective to minimize
  toxin decay coefficient: 0             # Toxin decay coefficient: None, INP or number
  decon decay coefficient: 0             # Decontaminant decay coefficient: None, INP or number
  feasible nodes: ALL                    # Feasible booster nodes
  infeasible nodes: NONE                 # Infeasible booster nodes
  max boosters: [2]                      # Maximum number of booster stations
  type: FLOWPACED                        # Booster source type: MASS or FLOWPACED
  strength: 4.0                          # Booster source strength [mg/min or mg/L depending on
                                         #   type]
  response time: 0.0                     # Time [min] between detection and booster injection
  duration: 600.0                        # Time [min] for booster injection
  evaluate: false                        # Evaluate booster placement: true or false,  default
                                         #   = false
solver:
  type: glpk                             # Solver type
  options:                               # A dictionary of solver options
  threads: 1                             # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
boostersim:
eventDetection:
boosterimpact:
```

Figure 7.4: The `booster_mip` configuration template file.

The amount of contaminant injected into the network for the contamination scenarios. If the type option is MASS, then the units for the strength are in mg/min. If the type option is CONCEN, FLOWPACED or SETPOINT, then units are in mg/L.

Required input unless a TSG or TSI file is specified.

species

The name of the contaminant species injected into the network. This is the name of a single species. It is required when using EPANET-MSX, since multiple species might be simulated, but only one is injected into the network. For cases where multiple contaminants are injected, a TSI file must be used.

Required input for EPANET-MSX unless a TSG or TSI file is specified.

start time

The injection start time that defines when the contaminant injection begins. The time is given in minutes and is measured from the start of the simulation. For example, a value of 60 represents an injection that starts at hour 1 of the simulation.

Required input unless a TSG or TSI file is specified.

end time

The injection end time that defines when the contaminant injection stops. The time is given in minutes and is measured from the start of the simulation. For example, a value of 120 represents an injection that ends at hour 2 of the simulation.

Required input unless a TSG or TSI file is specified.

tsg file

The name of the TSG scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSG file will override the location, type, strength, species, start and end times options specified in the WST configuration file. The TSG file format is documented in File Formats Section 13.12.

Optional input.

tsi file

The name of the TSI scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSI file will override the TSG file, as well as the location, type, strength, species, start and end time options specified in the WST configuration file. The TSI file format is documented in File Formats Section 13.13.

Optional input.

signals

Name of file or directory with information to generate or load signals. If a file is provided the list of inp tsg tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Optional input.

msx file

The name of the EPANET-MSX multi-species file that defines the multi-species reactions to be simulated using EPANET-MSX.

Required input for EPANET-MSX.

msx species

The name of the MSX species whose concentration profile will be saved by the EPANET-MSX simulation and used for later calculations.

Required input for EPANET-MSX.

merlion

A flag to indicate if the Merlion water quality simulator should be used. The options are true or false. If an MSX file is provided, EPANET-MSX will be used.

Required input, default = false.

booster mip

detection

The sensor network design used to detect contamination scenarios. The sensor locations are used to compute a detection time for each contamination scenario in the TSG file. The options are a list of EPANET node IDs or a file name which contains a list of EPANET node IDs.

Required input.

model type

The model type used to determine optimal booster station locations. Options include NEUTRAL (complete neutralization) or LIMIT (limiting reagent).

Required input, default = NEUTRAL.

model format

The modeling language used to build the formulation specified by the model type option. The options are AMPL and PYOMO. AMPL is a third party package that must be installed by the user if this option is specified. PYOMO is an open source software package that is distributed with WST.

Required input, default = PYOMO.

stoichiometric ratio

The stoichiometric ratio used by the limiting reagent model (LIMIT) represents the mass of disinfectant removed per mass of contaminant removed. The units for disinfectant mass and contaminant mass are determined by the type of injection used for each species (mg for chemical and CFU for biological). This can be a number or a list of numbers greater than 0.0. When a list is specified, the optimization will be performed for each number in this list. As the stoichiometric ratio approaches 0, the LIMIT model converges to the NEUTRAL model.

Required input if the model type = LIMIT.

objective

The impact metric used to place the booster stations. In the current version, all models support MC metric (mass of toxin consumed through the node demands). The PD metric is only supported in the LIMIT Pyomo model.

Required input, default = MC.

toxin decay coefficient

The contaminant (toxin) decay coefficient. The options are (1) None, which runs the simulations without first-order decay, (2) INP, which runs the simulations with first-order decay using the coefficient specified in the EPANET 2.00.12 INP file or (3) a number, which runs the simulation with first-order decay and the specified first-order decay coefficient in units of (1/min) (overrides the decay coefficient in the EPANET 2.00.12 INP file).

Required input, default = 0.

decon decay coefficient

The disinfectant (decontaminant) decay coefficient. The options are (1) None, which runs the simulations without first-order decay, (2) INP, which runs the simulations with first-order decay using the coefficient specified in the EPANET 2.00.12 INP file or (3) a number, which runs the simulation with first-order decay and the specified first-order decay coefficient in units of (1/min) (overrides the decay coefficient in the EPANET 2.00.12 INP file).

Required input, default = 0.

feasible nodes

A list that defines nodes that can be considered for the booster station placement problem.

The options are: (1) ALL, which specifies all nodes as feasible booster station locations; (2) NZD, which specifies all non-zero demand nodes as feasible booster station locations; (3) NONE, which specifies no nodes as feasible booster station locations; (4) a list of EPANET node IDs, which identifies specific nodes as feasible booster station locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as feasible booster station locations.

Required input, default = ALL.

infeasible nodes

A list that defines nodes that cannot be considered for the booster station placement problem. The options are: (1) ALL, which specifies all nodes as infeasible booster station locations; (2) NZD, which specifies non-zero demand nodes as infeasible booster station locations; (3) NONE, which specifies no nodes as infeasible booster station locations; (4) a list of EPANET node IDs, which identifies specific nodes as infeasible booster station locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as infeasible booster station locations.

Optional input, default = NONE.

max boosters

The maximum number of booster stations that can be placed in the network. The value must be a nonnegative integer or a list of nonnegative integers. When a list is specified, the optimization will be performed for each number in this list.

Required input.

type

The injection type for the disinfectant at the booster stations. The options are MASS or FLOWPACED. See the EPANET 2.00.12 user manual for additional information about source types Rossman (2000).

Required input.

strength

The amount of disinfectant injected into the network from the booster stations. If the source type option is MASS, then the units for the strength are in mg/min. If the source type option is FLOWPACED, then units are in mg/L.

Required input.

response time

The time in minutes between the detection of a contamination incident and the start of injecting disinfectants from the booster stations. The value is a nonnegative integer. For example, a value of 120 represents a 120 minutes or a 2 hour delay between the time of detection and the start of booster injections.

Required input.

duration

The length of time in minutes that disinfectant will be injected at the booster stations during the simulation. The value is a nonnegative integer. For example, a value of 240 means that a booster would simulate injection of disinfectant at a particular node for 4 hours. This duration is applied to all booster station locations identified in the optimization process.

Required input.

evaluate

The option to evaluate the booster station placement created from the optimization process. Optional input, default = false.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process.Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

In addition to these standard WST configuration options, the solver block can define specific options for the solver selected. The solver options should be modified according to the specific optimization problem. The `booster_mip` subcommand recognizes the following options in the solver block of the configuration file:

```
solver:
  type: glpsol
  options:
    mipgap: 0.01
```

The solver block above shows an example of using the public domain solver GLPK (glpsol) with the LP-file (lp) interface available in the modeling language Pyomo. A common option available with MIP solvers is mipgap, which is used to balance the quality of the solution found by the solver with the time taken to obtain the solution. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

7.2.4.3 Subcommand Output

The `booster_mip` subcommand creates a YAML file called <output prefix>booster_mip_output-<count>.yml (where <count> is an integer starting at 1) that contains an optimized list of node locations (EPANET node IDs) to inject the disinfectant, the final impact metric, the run date and CPU time. If more than one booster station design is requested in the WST configuration file, the <count> suffix is incrementally increased each time to create multiple YAML files. The log file called <output prefix>booster_msx_output.log contains basic debugging information. A visualization YAML configuration file named <output prefix>booster_mip_output_vis.yml is also created. The `visualization` subcommand is automatically run using this YAML file.

7.3 Booster Placement Subcommand Comparison

This section summarizes some of the major differences between the `booster_msx` and `booster_mip` subcommands. The table below lists some of the positives and negatives of each of the subcommands.

booster_msx vs booster_mip		
	Pros	Cons
booster_msx	Uses a more accurate representation of the reaction dynamics between the contaminant and the disinfectant. The user can specify higher order reactions.	Needs information on the specific contaminant that has been injected and its reaction dynamics with the disinfectant.
	Uses the various impact metrics available in sim2impact (e.g., MC, PD, PE, PK, EC) to identify booster station locations.	Solves the booster station placement problem with algorithms (EA and Network Solver), which are heuristics with non-provable optimality.
	Is not recommended for larger network models, since it could be very computationally intensive.	
booster_mip	Solves larger (less skeletonized) network models in reasonable time.	Requires using linear input-output water quality model (Merlion), which only supports first order decay for both contaminant and disinfectant. Furthermore, simplifying reaction assumptions are made in both Neutralization and Limiting Reagent formulations.
	Solves the booster station placement problem to provable optimality using the model assumptions.	Supports only one impact metric (MC) currently to identify booster station locations.
	Uses a generic stochiometric ratio for different classes of contaminant to model the reaction dynamics in case of limited information about the contaminant.	

7.4 Booster Placement Examples

Two booster station placement examples are provided. The first example determines booster station placement assuming complete inactivation of the contaminant (using NEUTRAL), and the second example evaluates this placement in terms of a more realistic reaction dynamic between the contaminant and the disinfectant (using MSX). The examples use the EPANET Example Network 3 INP file, Net3.inp. A contamination scenario ensemble is defined using all NZD nodes and a biological contaminant injection of 5.77e8 CFU/min (colony forming units per minute), starting at time 0 and continuing for 6 hours. Sensors located at nodes 15, 35, 219 and 253 are used to detect each contamination scenario and initiate the booster response action. Booster stations inject disinfectant at 4 mg/L for 12 hours after detection, since no additional response time is added between detection and booster station operation.

7.4.1 Example 1

The first example uses the booster_mip subcommand and the NEUTRAL approach. The model format is PYOMO and the solver is GLPK. These parameter options are listed in the configuration file, booster_mip_ex1.yml, shown in Figure 7.5. The maximum number of booster stations is listed as an array to indicate that five booster station designs should be created, using 2, 4, 6, 8 and 10 as the maximum number of booster stations to place in the network. This notation uses the generated model files to efficiently solve for more than one design. The feasible booster station locations are limited to NZD nodes.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: [NZD]
  type: MASS
  strength: 5.77e8
  start time: 0
  end time: 360
booster mip:
  detection: [’15’, ’35’, ’219’, ’253’]
  model type: NEUTRAL
  model format: PYOMO
  stoichiometric ratio: 0
  objective: MC
  toxin decay coefficient: 0
  decon decay coefficient: 0
  feasible nodes: NZD
  infeasible nodes: NONE
  fixed nodes: []
  max boosters: [2,4,6,8,10]
  type: FLOWPACED
  strength: 4
  response time: 0
  duration: 1440
  evaluate: false
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: ${CWD}/booster_mip_ex1/Net3
  debug: 0
```

Figure 7.5: The `booster_mip` configuration file for example 1.

The example can be executed using the following command line:

```
wst booster_mip booster_mip_ex1.yml
```

Since five booster station designs were requested, five YAML output files with the results were produced. The file Net3booster_mip_output_3.yml, shown below in Figure 7.6 contains results for placing six booster stations in the network.

The WST configuration file for example 1 can be modified for booster station placement using the LIMIT approach. The model type option is changed from NEUTRAL to LIMIT and the stoichiometric ratio is set to a value greater than 0. The stoichiometric ratio defines the unit mass of disinfectant needed to inactivate a unit mass of contaminant. For example, a stoichiometric ratio of 0.01 mg/CFU specifies that 0.01 mg of a disinfectant is needed to inactivate 1 CFU of contaminant.

```
# booster_mip output
general:
  version: 1.5                                       # WST version
  date: '2018-10-15'                                 # Run date
  cpu time: 27.142                                   # CPU time (sec)
  directory: C:/wst-1.5/examples/booster_mip_ex1
  log file: Net3booster_mip_output.log               # Log file
booster:
  nodes: ['101', '141', '171', '215', '219', '255']  # List of booster nodes
  objective: 73485746.8756287                        # Objective value
scenarios:
  injection node: null
  start time: null
  end time: null                                     # Injection node
  discarded:
    count: null                      # Number of discarded scenarios
  non-detected:
    count: null                      # Number of non-detected scenarios
    mass injected: null              # Mass injected (grams)
  detected:
    count: null                      # Number of detected scenarios
    detection time: null             # Detection time (min)
    mass consumed: null              # Mass consumed (grams)
    mass injected: null              # Mass injected (grams)
    pre-booster mass injected: null  # Mass injected before booster activate(grams)
    mass in tanks: null              # Mass remaining in tanks (grams)
    mass balence: null               # Mass balence
```

Figure 7.6: The booster_mip YAML output file for example 1.

7.4.2 Example 2

Since both the NEUTRAL and LIMIT formulations use simplifying assumptions to model the reaction dynamics between a contaminant and a disinfectant, it is often useful to evaluate a booster station design from the MIP methods using a more complex multi-species reaction model through the booster_msx subcommand. While the booster_msx subcommand coupled with an EPANET-MSX model can be used to optimize booster station placement, the number of function evaluations required for convergence often makes this process infeasible.

The multi-species reaction equations in the Net3_EColi_TSB.msx file describe the inactivation of $E.\ coli$ by chlorine and the reaction of $E.\ coli$ and chlorine with the nutrient broth (TSB) (Murray et al., 2011). The contamination scenarios are setup using the TSI file, Net3_EColi_TSB.tsi. This file defines the same $E.\ coli$ injection as in the NEUTRAL approach, but includes a TSB injection at all NZD nodes in the network as well. The configuration file, booster_msx_ex1.yml, is shown in Figure 7.7 to evaluate a booster station design from the NEUTRAL approach.

The example can be executed using the following command line:

```
wst booster_msx booster_msx_ex1.yml
```

This analysis indicates that the booster stations placed with the assumption that the disinfectant completely inactivates the contaminant underestimates the mass consumed given a more realistic disinfectant and contaminant reaction dynamics as represented in the the $E.\ coli$-TSB model. As expected, the booster_msx subcommand computes a higher mass consumed than the NEUTRAL or LIMIT models because of their simplifying reaction assumptions.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  tsi file: Net3/Net3_EColi_TSB.tsi
  msx file: Net3/Net3_EColi_TSB.msx
  msx species: EColi
impact:
  erd file: null
  metric: [MC]
  tai file: null
  response time: 0
  detection limit: [0.0]
  detection confidence: 1
  msx species: EColi
booster msx:
  detection: [’15’, ’35’, ’219’, ’253’]
  toxin species: EColi
  decon species: CL
  feasible nodes: [’101’,’141’,’171’,’215’,’219’,’255’]
  infeasible nodes: NONE
  max boosters: 6
  type: FLOWPACED
  strength: 4.0
  response time: 0.0
  duration: 720
solver:
  type: EVALUATE
  options: {}
  verbose: True
configure:
  output prefix: ${CWD}/booster_msx_ex1/Net3
  debug: 0
```

Figure 7.7: The `booster_msx` configuration file for example 2.

Chapter 8

Source Identification

If a contamination incident is detected by a water utility, it will be important to determine the time and location where the contaminant injection occurred. Once this information is available, the current extent of contamination within the network can be estimated and appropriate control and clean-up strategies can be devised in order to protect the population. The `inversion` subcommand included in WST is designed to calculate a list of possible injection nodes and times given a set of measurements that could come from manual grab samples and/or an event detection system (EDS) like CANARY (Hart and McKenna, 2012).

Three major challenges associated with source identification calculations are addressed by this subcommand.

- First, the source identification algorithms provided through this subcommand have to consider measurement data that can only provide a discrete yes/no indication of contamination at a particular sensor node and time. It is assumed that the current level of sensor technology can only provide standard water quality measures such as free chlorine, conductivity, pH, total organic carbon. Using these measurements, the EDS performs statistical analysis to detect anomalous changes from a baseline and provides a yes/no indication of potential contamination. Therefore, the source identification algorithms provided through this subcommand are designed to work with binary measurements.
- Second, measurement information available from a sparse set of fixed sensors might not be sufficient to narrow down the possible contamination nodes to a tractable number. One strategy is to get additional measurements in the form of manual grab samples from locations around the network to help the source identification calculations better identify the contamination location(s). The `grabsample` subcommand (described in Chapter 10) can be used to establish the best manual grab sample locations that will provide the most useful information to help establish the likely source(s). The source inversion algorithms in WST can use measurement information from both fixed sensors and manual grab samples.
- Third, during a contamination incident, being able to solve the source identification problem in real-time is of utmost importance. Therefore, all algorithms provided through this subcommand pay special attention to computational efficiency and typically perform source identification calculations within minutes on realistic large-scale networks.

A flowchart representation of the `inversion` subcommand is shown in Figure 8.1. The utility network model is defined by an EPANET 2.00.12 compatible network models (INP format) in WST. The sensor/EDS measurements are supplied through a measurements file (See File Formats Section 13.7). Additional details on the source inversion approach to identify the contaminant injection location(s) is supplied by the user in the WST configuration file.

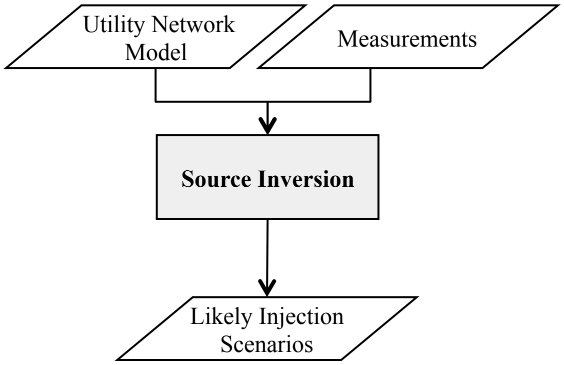


Figure 8.1: Contamination source identification flowchart.

8.1 Source Identification Formulations

The `inversion` subcommand contains three different source identification formulations, a Mixed Integer Programming (MIP) formulation, a formulation based on Bayesian probability calculations and a modified version of the Contaminant Status Algorithm by De Sanctis et al. (2009). The following subsections provide brief descriptions of these formulations.

8.1.1 MIP Formulations

Typically, optimization based methods try to find injection candidates that minimize the deviation between calculated values and the measurements (e.g., least squares), penalizing any mismatch above or below the measured values. The Mixed Integer Programming (MIP) formulation assumes that a field sensor (or manual grab sample) would yield a positive measurement if the contaminant concentration is above a certain positive threshold concentration and a negative measurement if it is below a certain negative threshold concentration. The objective function seeks to minimize the difference between the measured and calculated behavior according to this threshold. Therefore, if a sensor measurement (or a manual grab sample) yields a positive measurement, any corresponding calculated concentration from the water quality model above the positive threshold is deemed to be a perfect fit with this measurement data. Hence, when constructing an objective for estimation, only calculated concentrations below this positive threshold should be minimized. Likewise, if a sensor (or manual grab sample) yields a negative measurement, only the corresponding calculated concentration above the negative threshold should be minimized. Based on this idea, the base MIP formulation is presented below followed by descriptions of three additional variations that perform source inversion under different assumptions. For more detailed information, please refer to Mann et al. (2012b).

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(n,t)\in\mathbf{S}_-} \text{neg}_{n,t} + \sum_{(n,t)\in\mathbf{S}_+} \text{pos}_{n,t} && && (8.1)\\
\text{subject to} \quad & Gc_{n,t} = D\mathbf{m}_R && \forall n \in \mathbf{N}, t \in \mathbf{T} && (8.2)\\
& 0 \le m_{n,t} \le By_n && \forall n \in \mathbf{N}, t \in \mathbf{T} && (8.3)\\
& \sum_{n\in\mathbf{N}} y_n \le I_{\max} && y_n \in \{0,1\} && (8.4)\\
& \text{neg}_{n,t} \ge 0, \quad \text{neg}_{n,t} \ge c_{n,t} - \tau_{\text{neg}} && \forall\,(n,t) \in \mathbf{S}_- && (8.5)\\
& \text{pos}_{n,t} \ge 0, \quad \text{pos}_{n,t} \ge \tau_{\text{pos}} - c_{n,t} && \forall\,(n,t) \in \mathbf{S}_+. && (8.6)
\end{aligned}
$$

where N is the set of all nodes, T is the set of all time steps, S_- represents the set containing the node-time step pairs where the discrete measurement is a negative detection and S_{+} defines the set containing the node-time step pairs where the discrete measurement is a positive detection. The parameters G and D are matrices from the Merlion water quality model. The variable $c_{n,t}$ is the calculated concentrations from the water quality model at node n and time step t, m is the vector of unknown time-discretized contaminant injection profile over all node and time steps and $m_{n,t}$ is an element in the m_R vector representing unknown mass injected at node n and time step t. A binary variable, y_n, indicates contaminant injection at node n if $y_n{=}1$ and B is a reasonable upper bound on the contaminant injection mass flow rate $m_{n,t}$. The variable $I_{\max}$ is the maximum number of possible injection locations. The user supplies two thresholds, τ_{neg} and τ_{pos}, which can be used as concentration set-points indicating the presence or absence of contaminant. Having a gap between the positive and negative threshold provides users a (buffer) region of concentration values where really small fluctuations do not lead to a positive measurement. Therefore, the user can specify a higher positive threshold than the negative threshold to only flag significant concentration changes as positive detection. The variable $\text{neg}_{n,t}$ is the non-negative difference between the modeled concentration $c_{n,t}$ and the user supplied threshold τ_{neg} for node-time step pairs belonging to $\mathbf{S}_{-}$. The variable $\text{pos}_{n,t}$ is the non-negative difference between the modeled concentration $c_{n,t}$ and the user supplied threshold τ_{pos} for node-time step pairs belonging to $\mathbf{S}_{+}$.

Equation 8.1 is the MIP objective, which minimizes the mismatch between the discrete measurements and their corresponding concentrations calculated from the model given the detection thresholds. Equation 8.2 is the embedded linear water quality model (Merlion, see Section 12.1 for details). Equation 8.3 is the big-M constraint that enforces the bound on the maximum mass flow rate of the injections. Equation 8.4 is the maximum number of injections constraint, while Equation 8.5 and Equation 8.6 are part of the reformulation of the objective function to handle the threshold treatment discussed above. For further details please refer to Mann et al. (2012b). They are used to enforce $\text{neg}_{n,t}$ and $\text{pos}_{n,t}$ as non-negative differences between modeled concentration and threshold for positive and negative measurements, τ_{pos} and τ_{neg}, respectively.

This base MIP formulation for discrete measurements can be selected using the formulation option of MIP_discrete in the inversion block of the `inversion` WST configuration file.

The source inversion problem is ill-posed with non-unique solutions. To tackle this issue, the `inversion` subcommand solves the problem multiple times, each time adding additional feasibility constraints to exclude previously found solutions until the objective of the solution has reduced significantly. Therefore, the final result reported by the `inversion` subcommand contains a list of objective values for each solution and the corresponding source node that was identified for that particular solution.

The base MIP formulation allows for any type of injection profile including the ones shown in Figure 8.2. In the presence of sufficient measurement information, identification of any injection profile is reasonable. For instance, the Pulse profile in Figure 8.2 (shown in red), requires frequent measurements from the sensors in order to detect and characterize the injection. However, with relatively few fixed sensors and manual samples, identifying all possible injection profiles can be very challenging. Therefore, in the presence of limited information, some source identification methods only support continuous injections. To this aim, the `inversion` subcommand provides two additional restrictions that can be applied to the base MIP formulation. Depending on the possible contaminant injection profile assumptions, Figure 8.2 shows the different injection profile restrictions that are supported through the different formulation variations. The Step and No Decrease are two different injection profile restrictions that can be enforced by the formulation variations. The Pulse injection does not require additional constraints and is supported by the base formulation. When limited and/or less frequent measurement data is available (e.g., only manual grab samples), the Step or the No Decrease formulation variations are recommended.

The first variation of this formulation requires the injection to be a single step of any calculated strength S and can be run using the formulation option of MIP_discrete_step in the inversion block. The Step profile shown in Figure 8.2 illustrates an example of this type of injection. This variant is implemented by adding the following constraints to the base formulation:

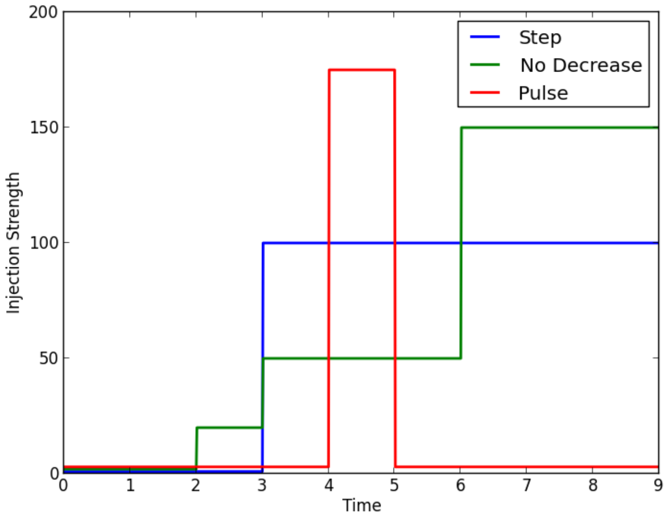


Figure 8.2: Three different types of contamination injection profiles.

$$m_{n,t} \leq S \qquad \forall n \in \mathbf{N}, t \in \mathbf{T} \tag{8.7}$$

$$m_{n,t} \geq S - B(1 - y_n) \qquad \forall n \in \mathbf{N}, t \in \mathbf{T} \tag{8.8}$$

The second variation adds constraint Equation 8.9 to the base formulation to allow for the case where the mass flow rate of the contaminant can stay the same or increase with time. The No Decrease profile shown in Figure 8.2 gives an example of this kind of injection. This variant of the formulation can be run using the formulation option of MIP_discrete_nd in the inversion block.

$$m_{n,t-1} \leq m_{n,t} \qquad \forall n \in \mathbf{N}, t \in \mathbf{T} : j \neq 0 \tag{8.9}$$

The third variation solves the Step formulation by fixing the binary variable y_n corresponding to a source node and solving the resulting linear program (LP) for all nodes $n \in, N$. The objective values for all LP solutions are compared and only a fraction of the identified nodes are reported in the results based on the candidate threshold option in inversion block of the configuration file. This LP variant can be run using the formulation option of LP_discrete in the inversion block.

8.1.2 Bayesian Probability Based Formulation

This formulation calculates the probability of a node being the true injection node using Bayes rule:

$$P(i|m) = \frac{P(m|i)P(i)}{P(m)} \tag{8.10}$$

where contamination incident i is an injection at a node and at a particular time step and $P(i|m)$ is the probability of an incident i given a set of measurements m. Here, $P(i)$ is the prior probability of an incident. This formulation assumes that only a single injection incident is possible, and therefore it uses an uniform prior of 1/(all possible incidents). Since it is difficult to estimate $P(m)$ (the prior probability of a measurement), this calculation is substituted by obtaining the $P(i|m)$ for all possible incidents and then normalizing them to 1. Finally, $P(m|i)$ is the probability of a measurement given an injection incident. It is calculated using the following equation:

$$P(m|i) = (1 - p_f)^{\mathrm{match}(i)} p_f^{\mathrm{meas} - \mathrm{match}(i)} \tag{8.11}$$

where, p_f is the probability of measurement failure, meas is the total number of measurements and $\mathrm{match}(i)$ is the number of discrete measurements that match the discrete concentrations obtained by simulating incident i. Note that calculating the discrete concentration profile obtained by simulating an incident requires a threshold that is specified by using the negative threshold option of the inversion block of the `inversion` configuration file.

After calculating the normalized probability $P(i|m)$ for all incidents, only those having a probability above a confidence limit are reported as the set of likely incidents along with their corresponding $P(i|m)$ values. This confidence limit is by default set to 95% (0.95) and can be changed by using the confidence option of the inversion block.

8.1.3 Contaminant Status Algorithm (CSA)

The Contaminant Status Algorithm (CSA), proposed by De Sanctis et al. (2009), performs source identification by assigning a status to each candidate node-time pair as either being safe (not an injection candidate), unsafe (possible injection candidate) or unknown. In WST, the CSA has been modified to assign a likeliness measure of 1 to a node if it is contained in the list of unsafe node-time pairs, while all other nodes are assigned a likeliness measure of 0.

CSA uses a linear input-output water quality model generated through the Particle Backtracking Approach (PBA) proposed by Shang et al. (2002). For every sensor j and analysis time t, this model provides the upstream reachability set, $U_j(t)$, that contains the list of node-time pairs that are hydraulically connected to that measurement. Using this set, a station source matrix, S_j, which represents the list of safe, unsafe and unknown nodes based on all the measurements available from sensor node j only, is updated iteratively using the following algorithm:

1. Initialize $S_j(i, \hat{t}) =$ Unknown, $\forall i \in \mathbf{N}, \forall \hat{t} \in \mathbf{T}$
2. For $(i, \hat{t}) \in U_j(t)$
   (a) For significant hydraulic connections (based on a threshold in the PBA input-output model)- if the current measurement at sensor j is positive, Set $S_j(i, \hat{t}) =$ Unsafe, else Set $S_j(i, \hat{t}) =$ Safe
   (b) For weak hydraulic connections (based on a threshold in the PBA input-output model), Set $S_j(i, \hat{t}) =$ Unknown

where N is the set of all candidate nodes and T is the set of all time steps in the time horizon. Based on the status from every station source matrix, S_j, a total source status matrix S that contains the overall status of all candidate node-time pairs, $(i, \hat{t})$, is updated using the following rules - an unsafe node-time pair can only change to safe based on its corresponding state in S_j; an unknown node-time pair can change to both safe or unsafe; and if a node-time pair is safe, it will remain safe. Hence, the total source status matrix is also updated iteratively over the complete list of measurement time steps to obtain the final status of all candidate injection node-time pairs. Consequently, CSA allows for multiple simultaneous injections, however, it assumes perfect measurements when marking candidate injections as safe.

8.2 Source Identification Solvers

The MIP algorithm builds an optimization formulation, and requires a MIP solver to perform source inversion. Therefore, if the MIP algorithm is selected (`algorithm: optimization`) as described in Section 8.3.2), then a solver needs to be specified. The solvers recognized by the `inversion` subcommand are the same as those recognized by `booster_mip` subcommand (See Section 7.2.3 for more details).

8.3 `inversion` Subcommand

The `inversion` subcommand is executed using the following command line:

```
wst inversion <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst inversion --help
```

8.3.1 Configuration File

The `inversion` subcommand generates a template configuration file using the following command line:

```
wst inversion --template <configfile>
```

The `inversion` template configuration file is shown in Figure 8.3. Brief descriptions of the options are included in the template after the # sign.

```
# inversion configuration template
network:
  epanet file: Net3.inp                  # EPANET 2.00.12 network file name
measurements:
  grab samples: measures.dat             # Measurements file name
inversion:
  algorithm: optimization                # Source inversion algorithm: optimization, bayesian
                                         #   or csa
  formulation: MIP_discrete_nd           # Optimization formulation type, optimization only
  model format: PYOMO                    # Source inversion optimization formulation: AMPL or
                                         #   PYOMO
  merlion water quality model: true      # Use Merlion water quality model for Bayesian
                                         #   algorithm
  horizon: 1440.0                        # Amount of past measurement data to use (min)
  num injections: 1.0                    # No. of possible injections
  measurement failure: 0.05              # Probability that a sensor fails
  positive threshold: 100.0              # Sensor threshold for positive contamination
                                         #   measurement
  negative threshold: 0.1                # Sensor threshold for negative contamination
                                         #   measurement
  feasible nodes: null                   # Feasible source nodes
  candidate threshold: null              # Objective cut-off for candidate nodes.
  confidence: null                       # Probability confidence for candidate nodes.
  output impact nodes: false             # Print likely injection nodes file
solver:
  type: glpk                             # Solver type
  options:                               # A dictionary of solver options
  threads: 1                             # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 8.3: The `inversion` configuration template file.

8.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `inversion` subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

measurements

grab samples

The name of the file that contains all the measurements from the manual grab samples and the fixed sensors. The measurement file format is documented in File Formats Section 13.7.

Required input.

inversion

algorithm

The algorithm used to perform source inversion. The options are optimization, bayesian, or csa. The optimization algorithm requires AMPL or PYOMO along with a MIP solver. The bayesian algorithm uses Bayes' Rule to update probability of a particular node being the contaminant source node. The CSA is the Contaminant Status Algorithm by De Sanctis et al. (2009).

Required input, default = optimization.

formulation

The formulation used by the optimization algorithm. The options are LP_discrete (discrete LP), MIP_discrete (discrete MIP), MIP_discrete_nd (discrete MIP with no decrease) or MIP_discrete_step (discrete MIP for step injection).

Required input for optimization algorithm, default = MIP_discrete.

model format

The modeling language used to build the formulation specified by the formulation option. The options are AMPL and PYOMO. AMPL is a third party package that must be installed by the user if this option is specified. PYOMO is an open source software package that is distributed with WST.

Required input for optimization algorithm, default = PYOMO.

merlion water quality model

This option is set to true to use the Merlion water quality model for simulating the candidate injections in the Bayesian probability-based method. It can be set to false to use EPANET 2.00.12 for these simulations. Note that the Merlion water quality model is required in either case to generate the initial list of candidate injections.

Optional input, default = true.

horizon

The minutes over which the past measurement data is used for source inversion. It is calculated backwards from the latest measurement time in the measurements file. All measurements in the measurements file that are within the horizon are used (both negative and positive). In the case of the CSA algorithm, the implementation assumes fixed sensors only, and all measurements at these sensors are assumed to be negative prior to the horizon. If the horizon is longer than the time between the latest measurement and simulation start time, then all the measurements are used for source inversion.

Required input, default = None (Start of simulation).

num injections

The number of possible injections to consider when performing inversion. Multiple injections are only supported by the MIP formulation. This value must be set to 1 for the LP model or the probability algorithm.

Required input for optimization algorithm, default = 1.

measurement failure

The probability that a sensors gives an incorrect reading. Must be between 0 and 1.

Required input for the Bayesian algorithm, default = 0.05.

positive threshold

The concentration threshold used by the sensors to flag a positive detection measurement. This is a parameter in the optimization algorithm (Equation 8.6).

Required input for optimization algorithm, default = 100 mg/L.

negative threshold

The concentration threshold used by the sensors to flag a negative detection measurement. This is a parameter in the optimization algorithm (Equation 8.5).

Required input for optimization algorithm, default = 0.0 mg/L.

feasible nodes

A list that defines nodes that can be considered for the source inversion problem. The options are: (1) ALL, which specifies all nodes as feasible source locations; (2) NZD, which specifies all non-zero demand nodes as feasible source locations; (3) a list of EPANET node IDs, which identifies specific nodes as feasible source locations; or (4) a filename, which is a space or comma separated file containing a list of specific nodes as feasible source locations.

Optional input.

candidate threshold

The objective cut-off value for candidate contamination incidents using the optimization algorithm. The objective value represents the likelihood of a particular node being the injection node (See Equation 8.13). The objective values are normalized to 1 and only the nodes having their objective values greater or equal to the threshold are reported in the inversion results.

Required input for optimization algorithm, default = 0.20.

confidence

The probability cut-off value for candidate contamination incidents using the Bayesian algorithm. The value is between 0 and 1.

Required for the Bayesian algorithm, default = 0.95.

output impact nodes

A option to output a Likely_Nodes.dat file that contains only the node IDs of the possible contaminant injection nodes obtained from the `inversion` subcommand. This file can be used as the feasible nodes for the next iteration of the `inversion` subcommand to only consider this set of possible contaminant injection nodes.

Optional input, default = false.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

8.3.3 Subcommand Output

The `inversion` subcommand creates several output files. The YAML file called <output prefix>inversion_output.yml contains a list of possible source node locations (EPANET node IDs), the associated objective or probability value (node likeliness) for each possible source node (a higher value indicates a higher likelihood of that node being the true contaminant injection node), the injection profiles (start and end times and the strength) for each possible source node, the run date and CPU time. The log file called <output prefix>inversion_output.log contains basic debugging information. The `inversion` subcommand also outputs an <output prefix>_profile.tsg file that contains the list of likely injection profiles in the TSG file format (See File Formats Section 13.12). As discussed in Chapter 10, this TSG file can be directly used as an input by the `grabsample` subcommand. A visualization YAML configuration file named <output prefix>inversion_output_vis.yml is also created. The `visualization` subcommand is automatically run using this YAML file.

8.4 Source Identification Examples

Three examples illustrating the use of the different source identification methods available in WST are presented. In the first example, the optimization formulation is used to solve a source identification problem, while the second example uses the Bayesian probability formulation and the third example uses the Contaminant Status Algorithm (CSA) to solve the exact same problem. An EPANET 2.00.12 network model (INP format) and a measurements file (See File Formats Section 13.7) are required to run the `inversion` subcommand.

Since real system data is not available, a measurements file required for the `inversion` subcommand can be generated using the `measuregen` executable (Executable Files Section 14.3). The three examples use the EPANET Example Network 3 input file (Net3.inp) as the network file, which runs a two day hydraulic and water quality simulation. An injection at node 151 is simulated from 8 hours until 24 hours, specified using the Net3_inversion.tsg file. The sensor locations are provided using the Net3_fixed_sensors file, and the measurements are obtained every 15 minutes with a concentration threshold of 0 indicating contamination. The following command line statement can be run from the examples folder to generate the measurements file:

```
measuregen --inp=Net3/Net3.inp --tsg=Net3/Net3_inversion.tsg --start-sensing-time=0
--stop-sensing-time=705 --measures-per-hour=4 --threshold=0 --ignore-merlion-warnings
--output-prefix=Net3/Net3 Net3/Net3_fixed_sensors
```

The resulting Net3_MEASURES.dat file is used in the following three examples.

8.4.1 Example 1

In the first example, the optimization method is used to solve the source identification problem. The configuration file, inversion_ex1.yml, shown in Figure 8.4 is used to identify the possible contaminant source locations. The MIP optimization formulation, MIP_discrete_step, is used for this example. The example uses the Net3_MEASURES.dat file generated using the `measuregen` executable.

```
network:
  epanet file: Net3/Net3.inp
measurements:
  grab samples: Net3/Net3_MEASURES.dat
inversion:
  algorithm: optimization
  formulation: MIP_discrete_step
  model format: PYOMO
  horizon: null
  num injections: 1.0
  measurement failure: 0.05
  positive threshold: 100.0
  negative threshold: 0.1
  feasible nodes: null
  candidate threshold: 0.25
  confidence: 0.95
  output impact nodes: false
solver:
  type: glpk
  options:
  logfile: null
  verbose: 0
  initial points: []
configure:
  output prefix: ${CWD}/inversion_ex1/Net3
  debug: 0
```

Figure 8.4: The `inversion` configuration file for example 1.

The example can be executed using the following command line:

```
wst inversion inversion_ex1.yml
```

The results are contained in the file Net3inversion_output.yml. A section of this results file is shown in Figure 8.5. The results contain a list of sets where each set contains - possible contaminant source node in the Nodes list (which contains a node Name and a Profile), CPU computation time in seconds and the Objective value corresponding to the solution which identifies that node as the source node. The Objective value for each candidate node n in the results file is related to the objective of the MIP formulations 8.1.1 (Equation 8.1). The objective calculated from the MIP formulation is transformed such that it is normalized to 1 and a higher value means a higher likelihood of a node being the source node. This transformation is done by the following equations:

$$\mathrm{INV\_NORM\_OBJ}_n = 1 - \frac{\mathrm{FORM\_OBJ}_n}{\max(\mathrm{FORM\_OBJ})} \qquad \forall n \in \mathbf{N} \tag{8.12}$$

$$\mathrm{Objective}_n = \frac{\mathrm{INV\_NORM\_OBJ}_n}{\max(\mathrm{INV\_NORM\_OBJ})} \qquad \forall n \in \mathbf{N} \tag{8.13}$$

where $\mathrm{FORM\_OBJ}_n$ (Formulation Objective) is the objective value as calculated from the MIP formulation Equation 8.1 when node n is identified as the most likely node, $\mathrm{INV\_NORM\_OBJ}_n$ is an intermediate variable that represents one (1) minus the normalized formulation objective and $\mathrm{Objective}_n$ is the normalized form of the $\mathrm{INV\_NORM\_OBJ}_n$ which is reported in the `inversion` subcommand results file.

This results file only contains the list of possible contaminant source nodes that have an objective (as calculated by 8.13) greater than the candidate threshold provided in the inversion block of the WST configuration file.

```
# inversion output
general:
  version: 1.5                                                # WST version
  date: ’2019-03-05’                                          # Run date
  cpu time: 260.199                                           # CPU time (sec)
  directory: C:/wst-1.5/examples/inversion_ex1
  log file: Net3inversion_output.log                          # Log file
inversion:
  tsg file: Net3profile.tsg
  likely nodes file: Net3Likely_Nodes.dat
  candidate nodes: [’149’, ’151’, ’153’, ’125’, ’123’, ’121’]  # List of candidate injection
                                                              #   nodes
  node likeliness: [1.0, 1.0, 1.0, 0.962, 0.298, 0.274]       # Likeliness measure of each
                                                              #   node being true injection
                                                              #   node.
```

Figure 8.5: The `inversion` YAML output file for example 1.

For this example scenario, the `inversion` subcommand is able to correctly identify node 151 as one of the three most likely source nodes. This means that given the current measurement information available, nodes 151, 153 and 149 are equally likely. Further measurements can be obtained from selected grab sampling locations that can help in distinguishing between these three potential source nodes. An example of how to use the `grabsample` subcommand to optimally select grab sampling location to improve the identification of the true contamination source is provided in the source identification case study in the Advanced Topics and Case Studies chapter 12.3.

8.4.2 Example 2

In this example, the Bayesian probability formulation is used to solve the same problem described in example 1. The configuration file, inversion_ex2.yml, shown in Figure 8.6, is used for this example. The bayesian formulation is selected by using the algorithm option in the inversion block.

```
network:
  epanet file: Net3/Net3.inp
measurements:
  grab samples: Net3/Net3_MEASURES.dat
inversion:
  algorithm: bayesian
  formulation: MIP_discrete_step
  model format: PYOMO
  horizon: null
  num injections: 1.0
  measurement failure: 0.05
  positive threshold: 100.0
  negative threshold: 0.1
  feasible nodes: null
  candidate threshold: 0.2
  confidence: 0.95
  output impact nodes: false
solver:
  type: glpk
  options:
  logfile: null
  verbose: 0
  initial points: []
configure:
  output prefix: ${CWD}/inversion_ex2/Net3
  debug: 0
```

Figure 8.6: The `inversion` configuration file for example 2.

The example can be executed using the following command line:

```
wst inversion inversion_ex2.yml
```

The results are contained in the file Net3inversion_output.yml shown in Figure 8.7. The likeliness value reported in this file corresponds to the probability value calculated by Equation 8.10. The probability algorithm is also able to correctly identify node 151 as the one of the three most probable source nodes along with nodes 149 and 153.

```
# inversion output
general:
  version: 1.5                                    # WST version
  date: ’2019-03-19’                              # Run date
  cpu time: 0.278                                 # CPU time (sec)
  directory: C:/wst-1.5/examples/inversion_ex2
  log file: Net3inversion_output.log              # Log file
inversion:
  tsg file: Net3profile.tsg
  likely nodes file: Net3Likely_Nodes.dat
  candidate nodes: [’149’, ’151’, ’153’]          # List of candidate injection nodes
  node likeliness: [0.333107, 0.333107, 0.333107] # Likeliness measure of each node being true
                                                  #   injection node.
```

Figure 8.7: The `inversion` YAML output file for example 2.

8.4.3 Example 3

In this example, the Contaminant Status Algorithm (CSA) is used to solve the same problem described in example 1. The configuration file, inversion_ex3.yml, shown in Figure 8.8, is used for this example. CSA is selected by using the algorithm option in the inversion block.

```
network:
  epanet file: Net3/Net3.inp
measurements:
  grab samples: Net3/Net3_MEASURES.dat
inversion:
  algorithm: csa
  formulation: MIP_discrete_nd
  model format: PYOMO
  horizon: 480.0
  num injections: 1.0
  measurement failure: 0.05
  positive threshold: 100.0
  negative threshold: 0.0
  feasible nodes: null
  candidate threshold: null
  confidence: null
  output impact nodes: false
solver:
  type: glpk
  options:
  logfile: null
  verbose: 0
  initial points: []
configure:
  output prefix: ${CWD}/inversion_ex3/Net3
  debug: 0
```

Figure 8.8: The `inversion` configuration file for example 3.

The example can be executed using the following command line:

```
wst inversion inversion_ex3.yml
```

The results are contained in the file Net3inversion_output.yml shown in Figure 8.9. Since in WST, a likeliness measure of 1 is assigned to a node if it is contained in the list of unsafe node-time pairs, while a likeliness measure of 0 is assigned to all other nodes, the node likeliness in the results output file is 1.0 for all possible contamination injection nodes. CSA is also able to correctly identify node 151 as the one of the four most probable source nodes along with nodes 125, 149 and 153.

```
# inversion output
general:
  version: 1.5                              # WST version
  date: ’2019-03-05’                        # Run date
  cpu time: 0.15                            # CPU time (sec)
  directory: C:/wst-1.5/examples/inversion_ex3
  log file: Net3inversion_output.log        # Log file
inversion:
  tsg file: Net3profile.tsg
  likely nodes file: Net3Likely_Nodes.dat
  candidate nodes: [’125’, ’149’, ’151’, ’153’]  # List of candidate injection nodes
  node likeliness: [1.0, 1.0, 1.0, 1.0]          # Likeliness measure of each node being true
                                                 #   injection node.
```

Figure 8.9: The `inversion` YAML output file for example 3.

Chapter 9

Uncertainty Quantification

Mathematical models used to simulate water distribution systems are subject to uncertainty. Effective characterization of uncertainty is critical for reliable analysis with simulation-based studies. Particularly for contamination incidents, uncertainty quantification is needed to effectively use simulation tools that provide insights into response actions. Hydraulic parameters that might cause uncertainty include: (1) customer demands at each node and time, (2) operational controls (e.g., valve settings, pump curves), (3) infrastructure topography and characteristics (e.g., missing pipes or junctions, effective pipe diameters) and (4) initial conditions (e.g., tank levels, pump statuses). Water quality parameters that might cause uncertainty include: (1) initial water quality, (2) contaminant species, (3) contaminant reaction dynamics, (4) the amount of contaminant injected, (5) injection location, (6) injection time and (7) injection duration.

The `uq` subcommand examines the effect of hydraulic and water quality uncertainty on the extent of contamination in terms of the identification of the contamination source. This subcommand can be used to quantify uncertainty after running source identification using the Bayesian probability based formulation (Section 8.1.2). Given a particular confidence level, nodes can be categorized according to their probability of contamination. Nodes whose contamination probability, γ_n, that are above a threshold are labeled with respect to their contamination state as LY for “likely yes,” LN for“likely no” and UN for “unknown.” For example, for a 95% confidence level:

$$\begin{cases} \gamma_n \geq 0.975 & \text{LY,} \\ 0.025 \leq \gamma_n \leq 0.975 & \text{UN,} \\ \gamma_n \leq 0.025 & \text{LN} \end{cases}$$

With the `uq` subcommand the effects of uncertainty in customer demand, isolation valve status, bulk reaction rate coefficient and contaminant injection location, start time, duration and rate can be studied on the size and location of the contamination incident.

A flowchart representation of the `uq` subcommand is shown in Figure 9.1. Given a list of EPANET 2.00.12 compatible network models (INP format) coupled with a list of injection scenarios (TSG format), the `uq` subcommand runs Monte Carlo simulations to estimate the probability that each node in the network is contaminated.

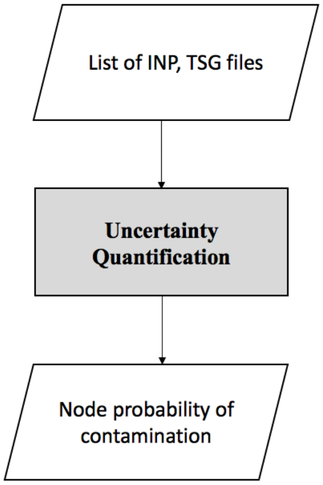


Figure 9.1: Uncertainty quantification flowchart.

9.1 Uncertainty Quantification Method

The uq subcommand runs an ensemble of scenarios where each scenario is defined using an INP and TSG file. The INP and TSG files contain the hydraulic and water quality parameters for the scenario. The results are used to compute the probability that each node is contaminated, using the following equation:

$$\gamma_n = \sum_{s \in S} \delta_{s,n} \beta_s, \tag{9.1}$$

where γ_n is the probability that node n is contaminated, β_s is the probability of scenario s and $\delta_{s,n}$ is a binary parameter that is 1 if scenario s contaminates node n, and 0 otherwise. The values of $\delta_{s,n}$ are determined from the simulations over the full potential scenario set, and the values for β_s are determined using equation (8.10) based on previous measurements provided in the measurements file. If no measurements file is provided, then the values for β_s are assumed to be uniform. A threshold value is used to decide whether a node is contaminated or not in a single scenario.

9.2 uq Subcommand

The uq subcommand is executed using the following command line:

```
wst uq <configfile>
```

where configfile is a WST configuration file in the YAML format.

The --help option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst uq --help
```

9.2.1 Configuration File

The uq subcommand generates a template configuration file using the following command line:

```
wst uq --template <configfile>
```

The uq template configuration file is shown in Figure 9.2. Brief descriptions of the options are included in the template after the # sign.

```
# uq configuration template
scenario:
  signals: list_scenarios               # Signal files, overrides TSG or TSI files
uq:
  analysis time: 60                     # Analysis time (min)
  threshold: 0.01                       # Contamination threshold. Default 0.01 (mg/L)
  confidence: 0.9                       # Probability confidence for candidate nodes (unitless).
measurements:
  grab samples: null                    # Measurements file name
configure:
  output prefix: ''                     # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                              # Debugging level, default = 0
```

Figure 9.2: The `uq` configuration template file.

9.2.2 Configuration Options

Full descriptions of the WST configuration options used by the `uq` subcommand are listed below.

scenario

signals

Name of file or directory with information to generate or load signals. If a file is provided the list of inp tsg tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Required input.

uq

analysis time

The time at which the manual grab sample should be taken. The algorithm determines the best possible manual grab sample location(s) based upon this time. Units: Minutes from the simulation start time in the EPANET 2.00.12 INP file.

threshold

This threshold determines whether or not an incident impacts a location (mg/L).

filter scenarios

This options enables filtering scenarios. Only those scenarios that match at least one of the measurements are considered in the optimal sampling analysis, default = False.

measurement failure

The probability that a sensor gives an incorrect reading. Must be between 0 and 1.

Required input for the Bayesian algorithm, default = 0.05.

confidence

The probability cut-off value for classifying nodes as certain to be contaminated, uncertain to be contaminated and certain to not be contaminated. The value is between 0 and 1. Nodes with probability greater than (1-confidence/2) are classified likely to be contaminated or like yes LY, nodes with probability less than confidence/2 are classified likely not contaminated LN and nodes with probability in between are uncertain nodes UN.

Required for node classification, default = 0.95 (unitless).

measurements

grab samples

The name of the file that contains all the measurements from the manual grab samples and the fixed sensors. The measurement file format is documented in File Formats Section 13.7.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

9.2.3 Subcommand Output

The uq subcommand creates two YAML files called <output prefix>_uq_scenarios.yml and <output prefix>_uq_nodes.yml that contains a list of probabilities for the scenarios and nodes, respectively. The log file named <output prefix>uq_output.log contains basic debugging information.

9.3 Uncertainty Quantification Example

A list of EPANET 2.00.12 INP and TSG files are required to run the uq subcommand. The configuration file for this example, uq_ex1.yml, is shown in Figure 9.3.

```
# uq configuration template
scenario:
  signals: Net3/uq/list_scenarios.dat    # Signal files, overrides TSG or TSI files
uq:
  analysis time: 60                      # Analysis time (min)
  threshold: 0.01                        # Contamination threshold. Default 0.01  (mg/L)
  confidence: 0.9                        # Probability confidence for candidate nodes. (unitless)
measurements:
  grab samples: null                     # Measurements file name
configure:
  output prefix: ${CWD}/uq_ex1/Net3      # Output file prefix
  output directory: ${CWD}/uq_ex1        # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 9.3: The uq configuration file for example 1.

The file with the list of scenarios for this example is shown in Figure 9.4

```
0  Net3/uq/inps/0.inp   Net3/uq/injections/0.tsg
1  Net3/uq/inps/1.inp   Net3/uq/injections/1.tsg
2  Net3/uq/inps/2.inp   Net3/uq/injections/2.tsg
```

Figure 9.4: List of scenarios

Summary information is printed to the screen, as shown in Figure 9.5.

The results from the uq subcommand can be represented in probability maps using the visualization subcommand (Chapter 11).

```
WST uq subcommand
---------------------------
Validating configuration file

NODE PROBABILITIES

Number of red nodes 0
Number of yellow nodes 47
Number of green nodes 50
+-------+-------------+----------------+
| Node  | Probability | Classification |
+=======+=============+================+
| 149   | 0.047101    | LN             |
+-------+-------------+----------------+
| 107   | 0.010870    | LN             |
+-------+-------------+----------------+
| 3     | 0.003623    | LN             |
+-------+-------------+----------------+
.
.
.
+-------+-------------+----------------+
| 189   | 0.097826    | UN             |
+-------+-------------+----------------+
| 195   | 0.054348    | UN             |
+-------+-------------+----------------+
| 184   | 0.115942    | UN             |
+-------+-------------+----------------+

WST normal termination
---------------------------
```

Figure 9.5: Screen output for example 1.

Chapter 10

Grab Sampling

When source identification is performed following initial detection of a contamination incident, it is likely that the identified set of possible injection locations is fairly large due to the limited measurement information available at the early stages of detection. As time progresses, more measurements become available to help decrease the number of possible injection locations. It is possible to obtain additional measurements in the form of grab samples from optimally selected locations that can help in quickly narrowing down the set of likely incident locations when source inversion calculations are performed again. The `grabsample` subcommand can be used to identify optimal grab sample locations that are likely to provide the most information in narrowing down the list of possible injection locations identified from the `inversion` subcommand.

A flowchart representation of the `grabsample` subcommand is shown in Figure 10.1. The required input for the `grabsample` subcommand includes a utility network model specified with an EPANET 2.00.12 compatible input file (INP) and a list of likely injection scenarios.

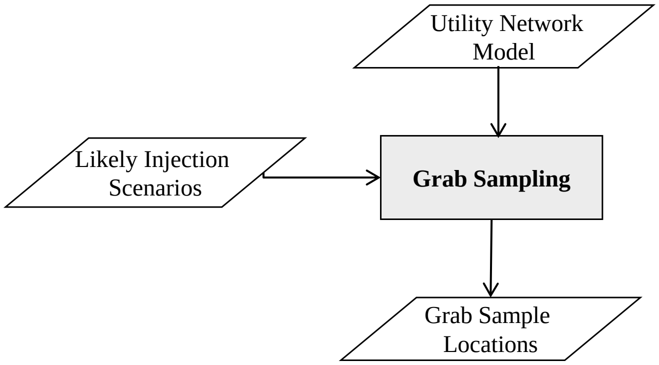


Figure 10.1: Grab sample flowchart.

10.1 Grab Sample Formulations

The `grabsample` subcommand contains three different grab sampling formulations, the distinguishability formulation and two probability-based formulations. The probability functions will likely be faster for larger problems (linear scaling), while the distinguishability formulation scales quadratically with the number of contamination scenarios. The following subsections provide brief descriptions of these formulations.

10.1.1 Distinguishability Formulation

Considering two possible contamination incidents i and j, if a particular sample location is impacted by incident i, but not impacted by incident j, then this sample location is able to distinguish between the two incidents. The `grabsample` subcommand can be used to identify grab sample locations that maximize the number of pairwise distinguishable incidents in a list of possible contamination incidents. The <output prefix>profile.tsg obtained from the `inversion` subcommand contains a list of possible injection locations. The data sets required by the optimization formulation below are obtained by simulating each possible incident using the EPANET 2.00.12 hydraulics model and either the EPANET 2.00.12 water quality model or the Merlion water quality model (Mann et al., 2012b), which can be selected using the merlion option in the scenario block of the configuration file.

The distinguishability problem formulation is:

$$\begin{aligned}
\text{maximize} \quad & \sum_{(i,j)\in PE} d_{ij} && && (10.1)\\
\text{subject to} \quad & \sum_{n\in D_{ij}} s_n \geq d_{ij} && \forall\,(i,j)\in PE && (10.2)\\
& \sum_{n\in G} s_n \leq S_{\max} + |F| && && (10.3)\\
& s_n \in \{0,1\} && \forall\, n \in G && (10.4)\\
& s_n = 1 && \forall\, n \in F && (10.5)\\
& 0 \leq d_{ij} \leq 1 && \forall\,(i,j)\in PE && (10.6)
\end{aligned}$$

where G is the set of all grab sample locations, F is the set of fixed sensor locations and PE is the pairwise set of all candidate incidents (i.e., possible contamination incidents). The variable D_{ij} is the set of sample locations that distinguish incident i from incident j, S_{max} is the maximum number of samples that can be taken at the same time (i.e., number of sampling teams), s_n is a binary variable that is 1 if node n is a good sample and 0 otherwise and d_{ij} is continuous variable that will be 1 if incident i is distinguishable from incident j and 0 otherwise.

Equation 10.1 represents the mixed-integer programming (MIP) objective which maximizes the number of pairwise distinguishable incidents. Equation 10.2 requires that at least one or more sample locations be selected for a distinguishable incident. Equation 10.3 limits the number of selected locations to be less than or equal to the number of sampling teams (specified by the user). Equation 10.4 defines s_n as a binary variable. Equation 10.5 ensures that the fixed sensor locations are always sampled since measurements from these fixed sensors are always available, which avoids double counting distinguished incidents. This formulation is the default formulation solved in the `grabsample` subcommand.

10.1.2 Probability-based Formulations

From a source inversion perspective, the contamination incident that agrees with the largest number of measurements is the contamination incident with the higher probability of occurrence. Similarly, all contamination incidents that disagree with many of the measurements have a low probability of occurrence. Following this idea, two optimization formulations were implemented in the `grabsample` subcommand in order to determine optimal sampling locations that are intended to maximize the probability of identifying the true contamination incident (or minimizing the probability of incidents that did not occur).

10.1.2.1 Maximization of expected number of scenarios that disagree with measurements

Given a set of potential contamination incidents, a few scenarios will agree with all the measurements, while many more will disagree. For this reason, the formulations in this section aim to select locations that maximize the number of disagreements between incidents and measurements (quickly reduce the probabilities of the incidents that are not likely to be consistent with observations). The development of an MILP problem formulation that meets this goal is presented next.

$$\begin{aligned}
\text{maximize} \quad & \sum_{s \in S} P_s^{\text{miss}} && && (10.7)\\
\text{subject to} \quad & P_s^{\text{miss}} = 1 - P_s^{\text{match}} && \forall\, s \in S && (10.8)\\
& P_s^{\text{match}} = \exp(\tilde{P}_s) && \forall\, s \in S && (10.9)\\
& \tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) && \forall\, s \in S && (10.10)\\
& \sum_{n \in N} x_n \leq S_{\max} && && (10.11)\\
& x_n \in \{0, 1\} && \forall\, n \in N. && (10.12)
\end{aligned}$$

Here P_s^{miss} is the probability that incident s disagrees with the outcome of the measurements at the selected locations. P_s^{match} (complement of P_s^{miss}) is given by the product of the probabilities $\alpha_{s,n}$ over all selected sampling locations,

$$P_s^{\text{match}} = \prod_{n \in N} \alpha_{s,n}^{x_n}. \qquad (10.13)$$

where x_n is a binary variable that will be 1 if location n is selected for sampling, and 0 otherwise. In the formulation, this product is written in equations (10.9) and (10.10). The parameter $\alpha_{s,n}$ is the probability that incident s disagrees with the outcome of a measurement taken at location n

$$\alpha_{s,n} = \begin{cases} \gamma_n & \text{if } \delta_{s,n} = 1; \\ 1 - \gamma_n & \text{otherwise.} \end{cases} \qquad (10.14)$$

where $\delta_{s,n}$ is a binary parameter that is 1 if incident s contaminates node n, and 0 otherwise. The values of $\delta_{s,n}$ are determined from the simulations pre-computed over the full potential incident set. The parameter γ_n is the probability that node n is contaminated and can be computed from the probability of the contamination incidents

$$\gamma_n = \sum_{s \in S} \delta_{s,n} \beta_s, \qquad (10.15)$$

Here β_s is the current estimate of the probability of contamination incident s. Finally, $S_{\max}$ is the maximum number of samples to be taken. The formulation as written is an MINLP because of Equation (10.9). However, it is easily made linear. Note that the equality in Equation (10.9) can be replaced with a lower bounding inequality. Since the objective function is maximizing P_s^{miss} (and pushing down on P_s^{match}), this inequality will always be satisfied with equality at the solution. Note also that this new inequality is convex and can be replaced with a set of linear under-estimators.

This new MILP formulation, referred to as problem Probability1, is shown below, where v_i are tangent points selected for the linear under-estimators of the exponential term and L is the set of indices corresponding to each of the linear under-estimators:

$$\begin{aligned}
\text{maximize} \quad & \sum_{s \in S} P_s^{\text{miss}} && && (10.16)\\
\text{subject to} \quad & P_s^{\text{miss}} = 1 - P_s^{\text{match}} && \forall\, s \in S && (10.17)\\
& P_s^{\text{match}} \geq \exp(v_i) + \exp(v_i)\left(\tilde{P}_s - v_i\right) && \forall\, i \in L,\ s \in S && (10.18)\\
& \tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) && \forall\, s \in S && (10.19)\\
& \sum_{n \in N} x_n \leq S_{\max} && && (10.20)\\
& x_n \in \{0,1\} && \forall\, n \in N, && (10.21)
\end{aligned}$$

10.1.2.2 Maximization of scenario with least number of measurement disagreements

A third formulation is also presented that maximizes the worst-case number of mismatches (instead of the expected value). This formulation does not contain the exponential term and is already an MILP without the need for any linear under-estimators, which avoids numerical issues that can occur when too many numerically similar under-estimators are added. This produces the max-min formulation shown below:

$$\begin{aligned}
\text{maximize} \quad \underset{s}{\text{minimize}} \quad & P_s^{\text{miss}} && \\
\text{subject to} \quad & P_s^{\text{miss}} = 1 - P_s^{\text{match}} && \forall\, s \in S\\
& P_s^{\text{match}} = \exp(\tilde{P}_s) && \forall\, s \in S\\
& \tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) && \forall\, s \in S\\
& \sum_{n \in N} x_n \leq S_{\max} && \\
& x_n \in \{0,1\} && \forall\, n \in N,
\end{aligned}$$

Recognizing that

$$\arg\min_s P_s^{\text{miss}} = \arg\min_s -P_s^{\text{match}}$$

and that

$$\arg\min_x -x = \arg\min_x -\exp(x),$$

the prior bilevel optimization formulation is reformulated to a single level optimization formulation as,

$$\begin{aligned}
\text{maximize} \quad & q && && (10.22)\\
\text{subject to} \quad & q \leq -\tilde{P}_s && && (10.23)\\
& \tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) && \forall\, s \in S && (10.24)\\
& \sum_{n \in N} x_n \leq S_{\max} && && (10.25)\\
& x_n \in \{0,1\} && \forall\, n \in N, && (10.26)
\end{aligned}$$

This formulation is referred to as Probability2, where q is an auxiliary variable that supports the max-min reformulation to a single level optimization problem.

10.2 Grab Sample Solvers

The grabsample subcommand requires standard MIP solvers to identify optimal grab sample locations. The solvers recognized by the grabsample subcommand are the same as those recognized by booster_mip subcommand (See Section 7.2.3 for more details).

10.3 grabsample Subcommand

The grabsample subcommand is executed using the following command line:

```
wst grabsample <configfile>
```

where configfile is a WST configuration file in the YAML format.

The --help option prints information about this subcommand.

```
wst grabsample --help
```

10.3.1 Configuration File

The grabsample subcommand generates a template configuration file using the following command line:

```
wst grabsample --template <configfile>
```

The grabsample template configuration file is shown in Figure 10.2. Brief descriptions of the options are included in the template after the # sign.

```
# grabsample configuration template
network:
  epanet file: Net3.inp                 # EPANET 2.00.12 network file name
scenario:
  location: null                        # Injection location: ALL, NZD or EPANET ID
  type: null                            # Injection type: MASS, CONCEN, FLOWPACED or SETPOINT
  strength: null                        # Injection strength [mg/min or mg/L depending on
                                        #   type]
  species: null                         # Injection species, required for EPANET-MSX
  start time: null                      # Injection start time [min]
  end time: null                        # Injection end time [min]
  tsg file: null                        # TSG file name, overrides injection parameters above
  tsi file: null                        # TSI file name, overrides TSG file
  signals: null                         # Signal files, overrides TSG or TSI files
  msx file: null                        # Multi-species extension file name
  msx species: null                     # MSX species to save
  merlion: false                        # Use Merlion as WQ simulator, true or false
grabsample:
  model format: PYOMO                   # Grab sample model format: AMPL or PYOMO
  sample criteria: distinguish          # Criteria to sample: distinguish, probability1,
                                        #   probability2
  sample time: 720.0                    # Sampling time (min)
  threshold: null                       # Contamination threshold. Default 0.001
  fixed sensors: null                   # Fixed sensor nodes
  nodes metric: null                    # Map of node to metric (e.g., EC, PI)
  list scenario ids: null               # List of scenario ids considered from the signals
                                        #   folder
  feasible nodes: null                  # Feasible sampling nodes
  num samples: null                     # Maximum number of grab samples, default = 1
  greedy selection: false               # Perform greedy selection. No optimization
  with weights: false                   # Perform optimization with weights in the objective
                                        #   function
  filter scenarios: false               # Filters scenarios that match measurements
measurements:
  grab samples: null                    # Measurements file name
solver:
  type: glpk                            # Solver type
  options:                              # A dictionary of solver options
  threads: 1                            # Number of concurrent threads or function evaluations
  logfile: null                         # Redirect solver output to a logfile
  verbose: 0                            # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                   # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                              # Debugging level, default = 0
```

Figure 10.2: The grabsample configuration template file.

The grabsample subcommand requires information about likely scenarios, which is set in the scenario block. These scenarios must be defined using a TSG file or by specifying the scenario location, type, strength, start and stop times (see Section 3.2 for more information on defining scenarios). In general, the TSG file created by the inversion subcommand will be used to define likely scenarios. Either the EPANET option or the Merlion option can be used as the water quality model, although, the Merlion water quality model is recommended for larger networks.

10.3.2 Configuration Options

Full descriptions of the WST configuration options used by the grabsample subcommand are listed below.

network

epanet file

The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

scenario

location

A list that describes the injection locations for the contamination scenarios. The options are: (1) ALL, which denotes all nodes (excluding tanks and reservoirs) as contamination injection locations; (2) NZD, which denotes all nodes with non-zero demands as contamination injection locations; or (3) an EPANET node ID, which identifies a node as the contamination injection location. This allows for an easy specification of single or multiple contamination scenarios.

Required input unless a TSG or TSI file is specified.

type

The injection type for the contamination scenarios. The options are MASS, CONCEN, FLOWPACED or SETPOINT. See the EPANET 2.00.12 user manual for additional information about source types (Rossman, 2000).

Required input unless a TSG or TSI file is specified.

strength

The amount of contaminant injected into the network for the contamination scenarios. If the type option is MASS, then the units for the strength are in mg/min. If the type option is CONCEN, FLOWPACED or SETPOINT, then units are in mg/L.

Required input unless a TSG or TSI file is specified.

species

The name of the contaminant species injected into the network. This is the name of a single species. It is required when using EPANET-MSX, since multiple species might be simulated, but only one is injected into the network. For cases where multiple contaminants are injected, a TSI file must be used.

Required input for EPANET-MSX unless a TSG or TSI file is specified.

start time

The injection start time that defines when the contaminant injection begins. The time is given in minutes and is measured from the start of the simulation. For example, a value of 60 represents an injection that starts at hour 1 of the simulation.

Required input unless a TSG or TSI file is specified.

end time

The injection end time that defines when the contaminant injection stops. The time is given in minutes and is measured from the start of the simulation. For example, a value of 120 represents an injection that ends at hour 2 of the simulation.

Required input unless a TSG or TSI file is specified.

tsg file

The name of the TSG scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSG file will override the location, type, strength, species, start and end times options specified in the WST configuration file. The TSG file format is documented in File Formats Section 13.12.

Optional input.

tsi file

The name of the TSI scenario file that defines the ensemble of contamination scenarios to be simulated. Specifying a TSI file will override the TSG file, as well as the location, type, strength, species, start and end time options specified in the WST configuration file. The TSI file format is documented in File Formats Section 13.13.

Optional input.

signals

Name of file or directory with information to generate or load signals. If a file is provided, the list of INP-TSG tuples will be simulated and the information stored in signals files. If a directory with the signals files is specified, the signal files will be read and loaded in memory. This input is only valid for the uq subcommand and the grabsample subcommand with probability based formulations.

Optional input.

msx file

The name of the EPANET-MSX multi-species file that defines the multi-species reactions to be simulated using EPANET-MSX.

Required input for EPANET-MSX.

msx species

The name of the MSX species whose concentration profile will be saved by the EPANET-MSX simulation and used for later calculations.

Required input for EPANET-MSX.

merlion

A flag to indicate if the Merlion water quality simulator should be used. The options are true or false. If an MSX file is provided, EPANET-MSX will be used.

Required input, default = false.

grabsample

model format

The modeling language used to build the formulation specified by the model format option. The options are AMPL and PYOMO. AMPL is a third party package that must be installed by the user if this option is specified. PYOMO is an open source software package that is distributed with WST.

Required input, default = PYOMO.

sample criteria

Determines which optimization model to solve. This option is only checked when running the problem with signal files. By default the optimization is based on distinguishability of pair-wise scenarios.

Optional input.

sample time

The time at which the manual grab sample should be taken. The algorithm determines the best possible manual grab sample location(s) based upon this time. Units: Minutes from the simulation start time in the EPANET 2.00.12 INP file.

Required input.

threshold

This threshold determines whether or not an incident impacts a candidate sample location.

Required input, default = 0.001.

fixed sensors

A list that defines nodes that are already fixed continuous sensor locations. The options are: (1) ALL, which specifies all nodes as fixed sensor locations; (2) NZD, which specifies non-zero demand nodes as fixed sensor locations; (3) NONE, which specifies no nodes as fixed sensor locations; (4) a list of EPANET node IDs, which identifies specific nodes as fixed sensor locations; or (5) a filename, which references a space or comma separated file containing a list of specific nodes as fixed sensor locations.

Optional input.

nodes metric

File containing a map of node to metric. The map is used for determining weighting factors in the objective of the distinguishability optimization formulation. Each line in the file has the node name separated by the corresponding metric.

Optional input.

list scenario ids

File containing list of scenarios to considered from the signals folder. Each line in the file has the signals ID and the contamination ID separated by a space. Optional input.

feasible nodes

A list that defines nodes that can be considered as potential sampling locations for the optimal sample location problem. The options are: (1) ALL, which specifies all nodes as feasible sampling locations; (2) NZD, which specifies all non-zero demand nodes as feasible sampling locations; (3) a list of EPANET node IDs, which identifies specific nodes as feasible sampling locations; or (4) a filename, which references a space or comma separated file containing a list of specific nodes as feasible sampling locations.

Optional input.

num samples

The maximum number of locations that can be sampled at one time. This is usually equal to the number of sampling teams that are available.

Required input, default = 1.

greedy selection

The option to select manual grab sample locations based upon a greedy search, which orders and selects the locations in order of the best solution. This does not require any optimization.

Optional input, default = false.

with weights

The option to add weights in the objective function of the distinguishability optimization formulation.

Optional input, default = false.

filter scenarios

This option enables filtering scenarios. Only those scenarios that match at least one of the measurements are considered in the optimal sampling analysis.

Optional input, default = false.

solver

type

The solver type. Each component of WST (e.g., sensor placement, flushing response, booster placement, source identification and grab sample) has different solvers available and more specific details are provided in the subcommand's chapter.

Required input.

options

A list of options associated with a specific solver type. More information on the options available for a specific solver is provided in the solver's documentation. The Getting Started Section 2.2 provides links to the different solvers.

Optional input.

threads

The maximum number of threads or function evaluations the solver is allowed to use. This option is not available to all solvers or all analyses.

Optional input.

logfile

The name of a file to output the results of the solver.

Optional input.

verbose

The solver verbosity level.

Optional input, default = 0 (lowest level).

initial points

nodes

A list of node locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing and booster_msx subcommands. This input causes an error for other subcommands.

Optional input.

pipes

A list of pipe locations (EPANET IDs) to begin the optimization process. Currently, this option is only supported for the network solver used in the flushing subcommand. This input causes an error for other subcommands.

Optional input.

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

10.3.3 Subcommand Output

The `grabsample` subcommand creates a YAML file called <output prefix>grabsample_output.yml that contains a list of node locations (EPANET node IDs) to take manual grab samples, the objective function value (based on the particular formulation selected), the run date and CPU time. The log file named <output prefix>grabsample_output.log contains basic debugging information. A visualization YAML configuration file named <output prefix>grabsample_output_vis.yml is also created, and following the execution of the `grabsample` subcommand, the `visualization` subcommand is automatically run using this YAML file.

10.4 Grab Sample Examples

Two examples for the `grabsample` subcommand are provided. The first example uses the distinguishability formulation, while the second uses the probability-based formulation, Probability1.

10.4.1 Example 1

An EPANET 2.00.12 network model (INP format) and a file containing a list of possible injection scenarios (e.g., a TSG file which is generated by the `inversion` subcommand) are required to run the `grabsample` subcommand. The configuration file for this example, grabsample_ex1.yml, is shown in Figure 10.3. The EPANET Example Network 3 input file, Net3.inp, is used for this example. The `grabsample` subcommand is typically used to identify sampling location after the results of the source identification calculation give a large list of candidate injection nodes. The time line of using the `inversion` and `grabsample` subcommand sequentially is provided in Figure 12.9. The TSG file, Net3_gs_profile.tsg, which contains the possible contamination incidents is created by the `inversion` subcommand using the measurement data created by the `measuregen` executable (Executable Files Section 14.3). For this example, the `measuregen` executable is used to simulate and obtain the measurements from a contaminant injection at node 251 at 24 hours. The injection is detected at 30.5 hours by using a set of fixed sensor locations defined in the Net3_fixed_sensors file. The list of eight equally likely contamination injection locations as listed in the TSG file, Net3_gs_profile.tsg, produced by the `inversion` subcommand is used as input to the `grabsample` subcommand along with the EPANET 2.00.12 network file. The sample time is set to 1890 minutes (31.5 hours), since it is assumed that it takes 60 minutes to perform source identification and obtain the manual grab samples (including travel time). The maximum number of manual grab samples that can be taken is two.

```
network:
  epanet file: Net3/Net3.inp
scenario:
  location: null
  type: null
  strength: null
  species: null
  start time: null
  end time: null
  tsg file: Net3/Net3_gs_profile.tsg
  tsi file: null
  msx file: null
  msx species: null
  merlion: true
grabsample:
  model format: PYOMO
  sample time: 1890.0
  threshold: null
  fixed sensors: Net3/Net3_fixed_sensors
  feasible nodes: null
  num samples: 2
  greedy selection: false
  with weights: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
  initial points: []
configure:
  output prefix: grabsample_ex1/Net3
  debug: 0
```

Figure 10.3: The `grabsample` configuration file for example 1.

The example can be executed using the following command:

```
wst grabsample grabsample_ex1.yml
```

The results are available in the Net3grabsample_output.yml, which is shown in Figure 10.4. The manual grab sample locations identified are nodes 241 and 251. Twenty-three pairwise incidents will be distinguished after taking the samples at these locations. To reiterate the configuration parameters, the sampling time is 1890 minutes and the maximum number of sampling locations is two. The grab sample locations identified in Figure 10.4 might be one of several solutions that produce the same objective value. If multiple grab sample locations provide the same ability to distinguish the contamination source, the solver will randomly pick a solution. Thus, the solution identified in Figure 10.4 could be different for other users.

```
# grabsample output
general:
  version: 1.5                         # WST version
  date: ’2019-03-19’                   # Run date
  cpu time: 0.351                      # CPU time (sec)
  directory: C:/wst-1.5/examples/grabsample_ex1
  log file: Net3grabsample_output.log  # Log file
grabsample:
  nodes: [’241’, ’251’]                # List of grabsample nodes
  objective: 96.0                      # Objective value
  threshold: null                      # Threshold
  count: 2                             # Count
  time: 1890.0                         # Time
```

Figure 10.4: The `grabsample` YAML output for example 1.

Next, as shown in Figure 12.9, the measurements from these selected grab sample locations (actual or simulated using `measuregen` executable) can be used to again perform source identification. Please refer to Section 12.3 for a complete case study of how to use the `inversion` and `grabsample` subcommands in tandem.

10.4.2 Example 2

In the second example, the probability-based formulation, Probability1, is used to select the optimal sampling locations, since the probability-based formulations are particularly efficient when the number of contamination scenarios is considerably large. The configuration file for this example, grabsample_ex2.yml, is shown in Figure 10.5.

```
# grabsample configuration template
network:
  epanet file: Net3/Net3.inp             # EPANET network file name
scenario:
  signals: Net3/grabsample/list_scenarios.dat   # Signal files, overrides TSG or TSI files
grabsample:
  model format: PYOMO                    # Grab sample model format: AMPL or PYOMO
  sample time: 720.0                     # Sampling time (min)
  threshold: null                        # Contamination threshold. Default 0.001
  fixed sensors: null                    # Fixed sensor nodes
  nodes metric: null                     # Map of node to metric (e.g., EC, PI)
  feasible nodes: null                   # Feasible sampling nodes
  num samples: 3                         # Maximum number of grab samples, default = 1
  sample criteria: probability1
  filter scenarios: True
measurements:
  grab samples: Net3/grabsample/MEAS.dat    # Measurements file name
solver:
  type: glpk                           # Solver type
  options:                               # A dictionary of solver options
  threads: 24                            # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: grabsample_ex2/Net3                    # Output file prefix
  output directory: grabsample_ex2       # Output directory
  debug: 0                               # Debugging level, default = 0
```

Figure 10.5: The `grabsample` configuration file for example 2.

The scenario information is provided with a list of pairs of INP-TSG files. This input allows the user to include different hydraulic and contamination in the set of potential scenarios. The list of potential scenarios used is shown in Figure 10.6. Ten different INP files with variations in the demand patterns are specified in the list to account for uncertainty in the hydraulics of the system. For simplicity a single TSG file is specified to provide information about the contamination scenarios. However, each entry in the list of scenarios could have a different TSG file.

```
0  Net3/grabsample/inps/0.inp  Net3/grabsample/injections.tsg
1  Net3/grabsample/inps/1.inp   Net3/grabsample/injections.tsg
2  Net3/grabsample/inps/2.inp   Net3/grabsample/injections.tsg
3  Net3/grabsample/inps/3.inp   Net3/grabsample/injections.tsg
4  Net3/grabsample/inps/4.inp   Net3/grabsample/injections.tsg
5  Net3/grabsample/inps/5.inp   Net3/grabsample/injections.tsg
6  Net3/grabsample/inps/6.inp   Net3/grabsample/injections.tsg
7  Net3/grabsample/inps/7.inp   Net3/grabsample/injections.tsg
8  Net3/grabsample/inps/8.inp   Net3/grabsample/injections.tsg
9  Net3/grabsample/inps/9.inp   Net3/grabsample/injections.tsg
```

Figure 10.6: List of scenarios example 2.

In addition, a list of the currently available measurements is provided in a measurement file with columns labeled as “location, time and measurement value.” The measurements are used to compute the probability of the scenarios following a Bayesian approach. When no measurements are provided, the probability distribution of the scenarios is assumed to be uniform. The measurement file in this example is shown in Figure 10.7

```
40  7200 1
111 7200 1
115 7200 1
```

Figure 10.7: List of measurements example 2.

The example can be executed using the following command:

```
wst grabsample grabsample_ex2.yml
```

The results are available in the Net3grabsample_output.yml, which is shown in Figure 10.8.

```
# grabsample output
general:
  version: 1.5                                              # WST version
  date: ’2019-03-05’                                        # Run date
  cpu time: 6.757                                           # CPU time (sec)
  directory: c:/wst-1.5/examples/grabsample_ex2             # Results directory
  log file: Net3grabsample_output.log                       # Log file
grabsample:
  nodes: [’50’, ’115’, ’3’]                                 # List of grabsample nodes
  objective: 9.193169619999995                              # Objective value
  threshold: null                                           # Threshold
  count: 3                                                  # Count
  time: 720.0                                               # Time
```

Figure 10.8: The `grabsample` YAML output for example 2.

Chapter 11

Visualization

Visualization tools are an important aspect of network analysis. For example, after completing a sensor placement optimization, it is important to understand how the physical location of the sensors relates to the underlying structure of the water distribution network. The `visualization` subcommand overlays graphical layers on a water distribution network and creates a HyperText markup language (HTML) file with scalar vector graphics that can be opened in a Web browser. The HTML file provides an interactive visualization of the results from which images can be saved for later use via a screen capture (saved as a JPEG, PNG). After the HTML file is opened in a Web browser, the user has the ability to (1) scroll over node and link elements to identify the respective EPANET ID, (2) scroll over the legend to isolate a specific layer, (3) move the legend and (4) zoom or pan the screen to change the size and location of the network.

The `visualization` subcommand includes an extensive number of graphic options within the configuration file. To format the appearance of the network model, the user can define the color, size and opacity for the network elements (e.g., junctions, reservoirs, tanks, pipes, pumps and valves). The user can also decide which network elements to include in the legend. Multiple node and link layers can then overlay the network model. The order of those layers is defined by the user. For each layer, the user can select the layer shape (for node layers) along with the color, size and opacity. The color, size and opacity can be defined as a constant or can be set as a function of the layers value. These options can be set independently for the layers fill and line. Each layer is assigned a label to be used in the legend. Other options include the screen size and background color, and the legend location and background color.

Several WST subcommands automatically run the `visualization` subcommand upon completion to generates graphical representation of the results. These include `sp`, `flushing`, `booster_msx`, `booster_mip`, `inversion` and `grabsample`. The graphic can be modified by editing the `visualization` configuration file, which is also automatically generated, and rerunning the `visualization` subcommand. A flowchart representation of the `visualization` subcommand is shown in Figure 11.1. The utility network model is defined by an EPANET 2.00.12 compatible network model (INP format). Graphic options are supplied through the `visualization` WST configuration file.

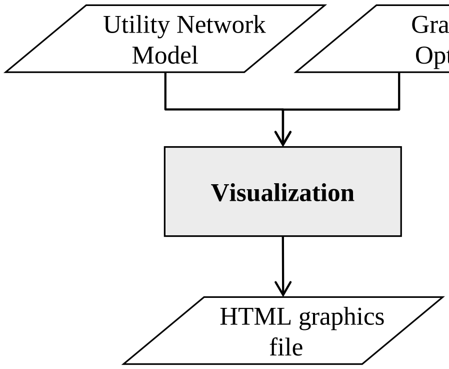


Figure 11.1: Visualization flowchart.

11.1 Color and Shape Options

The color of a network element is specified using a six character hexadecimal (HEX) color code or using a predefined color name. HEX color codes can be found at various website, including `http://www.color-hex.com/color-wheel/`, and can be used to create any color between black (#000000) and white (#FFFFFF). The following predefined colors can also be used in the the `visualization` subcommand.

```
  Name    RGB           HEX
============================================
  red     [225,0,0]     \#FF0000
  orange  [225,165,0]   \#FFA500
  yellow  [225,225,0]   \#FFFF00
  green   [0,128,0]     \#008000
  blue    [0,0,225]     \#0000FF
  purple  [128,0,128]   \#800080
  black   [0,0,0]       \#000000
  white   [225,225,225] \#FFFFFF
  lime    [0,225,0]     \#00FF00
  navy    [0,0,128]     \#000080
  aqua    [0,225,225]   \#00FFFF
  teal    [0,128,128]   \#008080
  olive   [128,128,0]   \#808000
  maroon  [128,0,0]     \#800000
  fuchsia [225,0,225]   \#FF00FF
  silver  [192,192,192] \#C0C0C0
  gray    [128,128,128] \#808080
```

The shape of a network element is specified using one of the following predefined shapes, using either the long or short name.

```
  Long Name    Short Name
===========================
  circle        o
  square        s
  triangle      t
  diamond       d
  plus          +
  x             x
```

11.2 Data from YAML Files

Within the `visualization` WST configuration file, layers can be defined by (1) directly including the network element values, or (2) referencing data in an external YAML file.

The following subset of a `visualization` WST configuration file demonstrates how element values are directly included in the layers block.

```
layers:
  locations: [’115’, ’101’, ’171’]
  file: null
```

The same data can be stored in an external YAML file and referenced in the `visualization` WST configuration file, as shown below.

```
layers:
  locations: ’["flushing"]["nodes"][i]’
  file: data.yml
```

In this example, the referenced file, data.yml, contains the following information.

```
flushing:
  nodes: [’115’, ’101’, ’171’]
```

The WST subcommands (e.g., `sp`, `flushing`) that automatically run the `visualization` subcommand upon completion read data from external YAML files.

11.3 `visualization` Subcommand

The `visualization` subcommand is executed using the following command line:

```
wst visualization <configfile>
```

where `configfile` is a WST configuration file in the YAML format.

The `--help` option prints information about this subcommand, such as usage, arguments and a brief description:

```
wst visualization --help
```

11.3.1 Configuration File

The `visualization` subcommand generates a template configuration file using the following command line:

```
wst visualization --template <configfile>
```

The `visualization` WST template configuration file is shown in Figure 11.2. Brief descriptions of the options are included in the template after the # sign.

```
# visualization configuration template
network:
  epanet file: Net3.inp                   # EPANET 2.00.12 network file name
visualization:
  screen:
    color: white                          # Screen color, HEX or predefined code
    size: [1200, 800]                     # Screen size [width, height] in pixels
  legend:
    color: white                          # Legend color, HEX or predefined code
    scale: 1.0                            # Legend text size multiplier, real number
    location: [800, 20]                   # Legend location [left, bottom] in pixels
  nodes:
    color: null                           # Node color, HEX or predefined code
    size: null                            # Node size, real number
    opacity: 0.6                          # Node opacity, real number
  links:
    color: null                           # Link color, HEX or predefined code
    size: null                            # Link size, real number
    opacity: 0.6                          # Link opacity, real number
  layers:
    -
      label: pipes with different colors  # Label used in legend
      locations: [’101’, ’171’]           # Data locations, list of EPANET IDs
      file: null                          # Locations from file, YAML format
      location type: link                 # Location type, node or link
      shape: [circle]                     # Marker shape, predefined name
      fill:
        color: [yellow, red]              # Fill color, HEX or predefined code
        size: [10, 20]                    # Fill size, real number
        opacity: [0.5, 1]                 # Fill opacity, real number
        color range: null                 # Fill color range [min, max]
        size range: null                  # Fill size range [min, max]
        opacity range: null               # Fill opacity range [min, max]
      line:
        color: null                       # Line color, HEX or predefined code
        size: null                        # Line size, real number
        opacity: 0.6                      # Line opacity, real number
        color range: null                 # Line color range [min, max]
        size range: null                  # Line size range [min, max]
        opacity range: null               # Line opacity range [min, max]
    -
      label: orange nodes                 # Label used in legend
      locations: [’105’, ’35’, ’15’]      # Data locations, list of EPANET IDs
      file: null                          # Locations from file, YAML format
      location type: node                 # Location type, node or link
      shape: [diamond]                    # Marker shape, predefined name
      fill:
        color: orange                     # Fill color, HEX or predefined code
        size: 10                          # Fill size, real number
        opacity: 0.6                      # Fill opacity, real number
        color range: null                 # Fill color range [min, max]
        size range: null                  # Fill size range [min, max]
        opacity range: null               # Fill opacity range [min, max]
      line:
        color: black                      # Line color, HEX or predefined code
        size: 1                           # Line size, real number
        opacity: 1                        # Line opacity, real number
        color range: null                 # Line color range [min, max]
        size range: null                  # Line size range [min, max]
        opacity range: null               # Line opacity range [min, max]
configure:
  output prefix: Net3                     # Output file prefix
  output directory: SIGNALS_DATA_FOLDER   # Output directory
  debug: 0                                # Debugging level, default = 0
```

Figure 11.2: The `visualization` configuration template file.

11.3.2 Configuration Options

Full descriptions of the WST configuration options used by the `visualization` subcommand are listed below.

network

epanet file
The name of the EPANET 2.00.12 input (INP) file that defines the water distribution network model.

Required input.

visualization

screen

color
The screen background color defined using a HEX color code or predefined color name.

Optional input, default = white

size
The screen size [width, height] in pixels.

Optional input, default = [1000,600]

legend

color
The legend background color defined using a HEX color code or predefined color name.

Optional input, default = white

scale
The legend text size multiplier, real number.

Optional input, default = 1.0

location
The legend location [left, top] in pixels.

Optional input, default = [10,10] (upper left)

nodes

color
The node color defined using HEX color code or predefined color name. The color will apply to junctions, reservoirs and tanks. If the color is left blank, then junctions are black, reservoirs are blue and tanks are green.

Optional input, default = None

size
The node size, real number.

Optional input.

opacity
The node opacity, real number between 0.0 (transparent) and 1.0 (opaque).

Optional input, default = 0.6

links

color
The link color defined using HEX color code or predefined color name. The color will apply to pipes, pumps and valves. If the color is left blank, then pipes are black, pumps are yellow and valves are turquoise.

Optional input, default = None

size

The link size, real number.

Optional input.

opacity

The link opacity, real number between 0.0 (transparent) and 1.0 (opaque).

Optional input, default = 0.6

layers

label

The layer label used in the legend.

Optional input, default = None

locations

The data locations to plot over the network. Locations are specified using a list of EPANET IDs.

Required input unless an external file is specified.

file

The name of an external file that contains data to be used in the visualization. The file is in YAML format.

Required input unless 'locations' are specified. Data from a file overrides data specified in 'locations'

location type

The location type is used to indicate if the EPANET ID is of type 'node' (junction, reservoir, tank) or 'link' (pipe, pump, valve).

Optional input. If left blank, node is tested before link

shape

The marker shape, used only for node type layers. The shape can be a single string, or a list of strings which is the same length as the location data.

Optional input, only used for node type layers, default = circle

fill

color

The fill color defined using HEX color code or predefined color name.

Optional input, default = None

size

The fill size, real number.

Optional input.

opacity

The fill opacity, real number between 0.0 (transparent) and 1.0 (opaque).

Optional input, default = 0.6

color range

The fill color range used to scale line data.

Optional input, default = [data min, data max]

size range

The fill size range used to scale line data.

Optional input, default = [data min, data max]

opacity range

The fill opacity range used to scale line data.

Optional input, default = [data min, data max]

line

color

The line color defined using HEX color code or predefined color name.

Optional input, default = None

size

The line size, real number.

Optional input.

opacity

The fill opacity, real number between 0.0 (transparent) and 1.0 (opaque).

Optional input, default = 0.6

color range

The line color range used to scale line data.

Optional input, default = [data min, data max]

size range

The line size range used to scale line data.

Optional input, default = [data min, data max]

opacity range

The line opacity range used to scale line data.

Optional input, default = [data min, data max]

configure

output prefix

The prefix used for all output files.

Required input.

output directory

The output directory to store the results.

debug

The debugging level (0 or 1) that indicates the amount of debugging information printed to the screen, log file and output yml file.

Optional input, default = 0 (lowest level).

For additional control over the way that junctions, reservoir, tanks, pipes, pumps and valves are displayed, the following YAML blocks can be added to the visualization configuration block. The color, size and opacity of each element can be changed; and the element can be added to the legend. These options will override the node and link blocks.

```
junctions:
  color: red
  size: 5.0
  opacity: 0.5
reservoirs:
  color: orange
  size: 12.0
  opacity: 1
tanks:
  color: yellow
  size: 12.0
  opacity: 1
pipes:
  color: green
```

```
    size: 3.0
    opacity: 0.5
  pumps:
    color: blue
    size: 3.0
    opacity: 1
  valves:
    color: purple
    size: 3.0
    opacity: 0.7
```

11.3.3 Subcommand Output

The `visualization` subcommand creates a HTML file named <output prefix>visualization_output.html that contains scalar vector graphics that can be opened in a Web browser. Two other files are created: (1) an output YAML file named <output prefix>visualization_output.yml that includes run date and CPU time, and (2) a log file named <output prefix>visualization_output.log that includes basic debugging information.

11.4 Visualization Examples

An EPANET 2.00.12 network model input file (INP format) and a configuration file, which contain the graphics options, are required to run the `visualization` subcommand. Several examples for visualization are given below.

11.4.1 Example 1

The first example customizes the color, size and opacity of the network elements (junctions, reservoirs, tanks, pipes and pumps). The reservoir and tank colors are specified using HEX color codes. Additionally, the graphic highlights five pipes and uses their diameter to scale the pipe width. All the data is supplied in the configuration file. The configuration file, visualization_ex1.yml, for this example is shown in Figure 11.3.

The example can be executed using the following command line:

```
wst visualization visualization_ex1.yml
```

The resulting graphic is shown in Figure 11.4.

```
network:
    epanet file: Net3/Net3.inp
visualization:
  screen:
    color: white
    size: [1600, 1000]
  legend:
    color: white
    scale: 2.0
    location: [20, 20]
  junctions:
    color: black
    size: 5.0
    opacity: 0.5
  reservoirs:
    color: ’#a0b0f8’
    size: 18.0
    opacity: 1
  tanks:
    color: ’#3ec427’
    size: 18.0
    opacity: 1
  pipes:
    color: black
    size: 1.0
    opacity: 0.5
  pumps:
    color: maroon
    size: 10.0
    opacity: 1
  layers:
    label: Five Pipes
    locations: [’145’, ’287’, ’155’, ’231’, ’175’]
    file: null
    location type: link
    shape: circle
    fill:
      color: red
      size: [8, 10, 12, 24, 30]
      opacity: 1
      color range: null
      size range: null
      opacity range: null
configure:
  output prefix: ${CWD}/visualization_ex1/Net3
  debug: 0
```

Figure 11.3: The `visualization` configuration file for example 1.

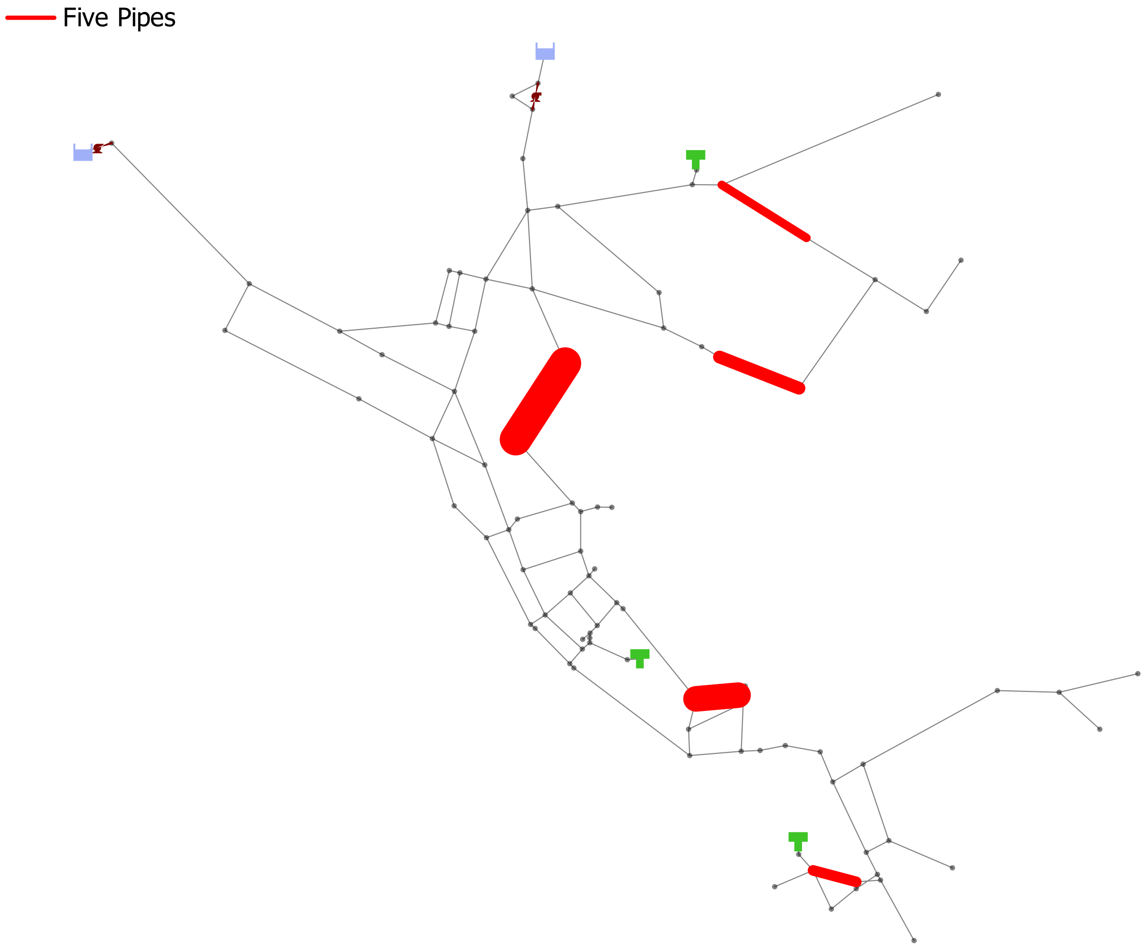


Figure 11.4: Graphic from `visualization` example 1.

11.4.2 Example 2

The second example uses an external data file to define locations and values to be used in the network graphic. The configuration file, visualization_ex2.yml, for this example is shown in Figure 11.5. The location file used in this example is shown in Figure 11.6. This example uses the pipe length to scale the size and opacity of the links and the base demand to scale the color and size of the nodes. This graphic shows that link 329 is very long compared to the other 30-inch diameter pipes and that node 109 has the largest base demand. The size range option in the configuration file is used to automatically scale the size and opacity of each layer. For example, the link length ranges from 1 to 45,500, but is scaled to a size range of 5 to 20.

```
network:
    epanet file: Net3/Net3.inp
visualization:
  screen:
    color: white
    size: [1600, 1000]
  legend:
    color: white
    scale: 2.0
    location: [20, 20]
  layers:
  - label: 30 inch diameter pipes
    locations: ’["links"][i]’
    file: Net3/Net3_locations.yml
    location type: link
    shape: circle
    fill:
      color: blue
      size: ’["length"][i]’
      opacity: ’["length"][i]’
      color range: null
      size range: [5,20]
      opacity range: [0.5,1]
  - label: Nodes with base demand > 100
    locations: ’["nodes"][i]’
    file: Net3/Net3_locations.yml
    location type: node
    shape: circle
    fill:
      color: ’["base demand"][i]’
      size: ’["base demand"][i]’
      opacity: 1
      color range: [orange, red]
      size range: [15,35]
      opacity range: null
configure:
  output prefix: ${CWD}/visualization_ex2/Net3
  debug: 0
```

Figure 11.5: The `visualization` configuration file for example 2.

The example can be executed using the following command line:

```
wst visualization visualization_ex2.yml
```

The resulting graphic is shown in Figure 11.7.

```
nodes: [109, 101, 119, 151, 111, 105, 103, 199, 117, 189]
base demand: [231.4, 189.95, 176.13, 144.48, 141.94, 135.37, 133.2,
  119.32, 117.71, 107.92]
links: [123, 125, 173, 175, 177, 179, 183, 187, 189, 321, 329, 330, 333]
length: [2000, 1500, 2080, 2910, 2000, 430, 590, 1270, 50, 1200, 45500, 1, 1]
```

Figure 11.6: The location file used in `visualization` example 2.

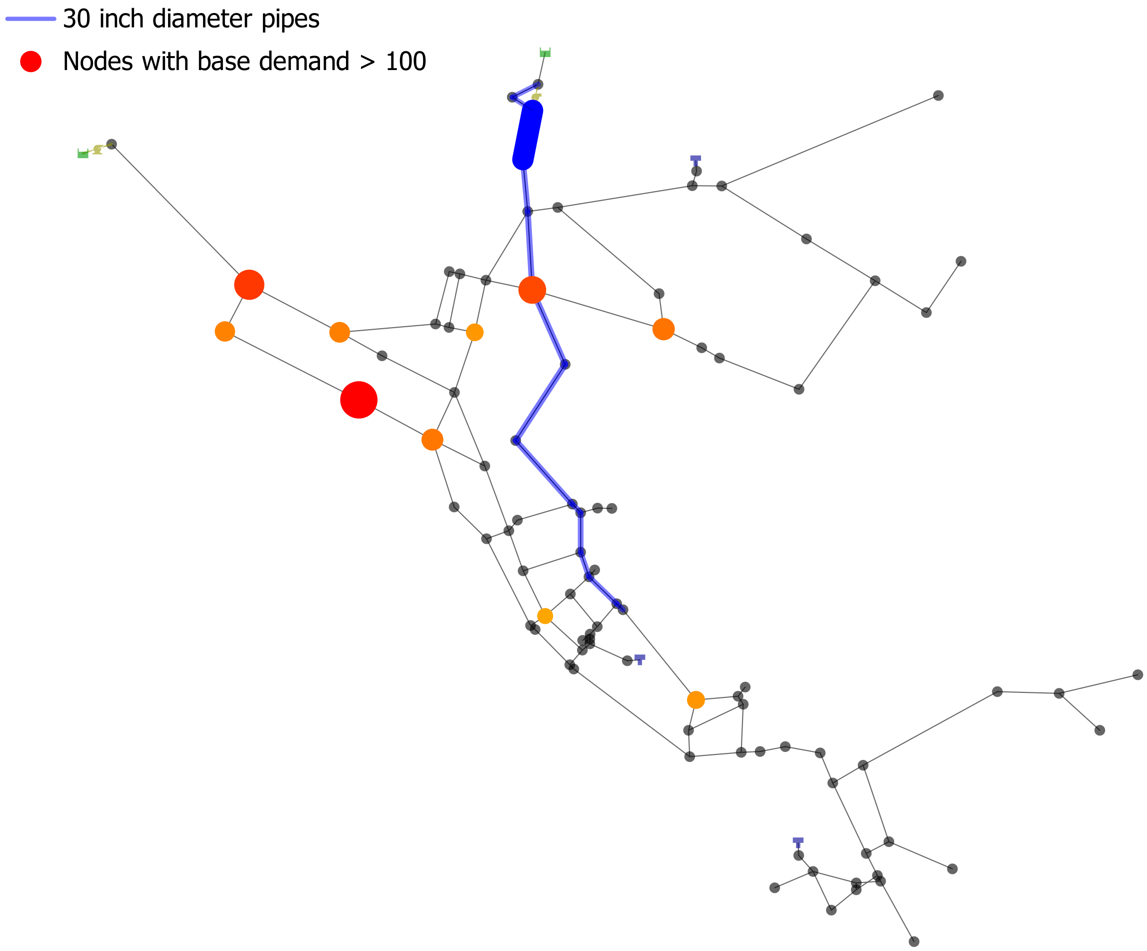


Figure 11.7: Graphic from `visualization` example 2.

Chapter 12

Advanced Topics and Case Studies

This chapter provides more background information on the Merlion water quality model and discusses a few of the more advanced topics for the sensor placement problem. In addition, a few case study applications using the different WST subcommands are provided.

12.1 Merlion Water Quality Model

The Merlion water quality modeling framework is provided with WST to enable fast multi-scenario simulations and solution of optimization problems that require an embedded water quality model (e.g., booster placement with `booster_mip`, source identification MIP formulation). In this section, the model equations and the calculations performed inside WST in order to generate a linear system of equations to describe the water quality in a network are briefly described. The equations and the discretization process described in this section do not require any additional work from the user (except for selecting the merlion option in the configuration file). More details about the Merlion modeling framework is provided in (Mann et al., 2012a).

The model formulation ensures mass balances at all junctions, pipes and tanks. The following mass balance equations describe the transport of a species inside the network. For simplicity, complete instantaneous mixing is assumed for the tanks, and plug flow is assumed for the pipes.

$$c_n(t) = \frac{\sum_{i \in \Gamma_n^O(t)} Q_i(t)\hat{c}_i^O(t) - \sum_{i \in \Gamma_n^I(t)} Q_i(t)\hat{c}_i^I(t) + m_n(t)}{\sum_{i \in \Gamma_n^O(t)} Q_i(t) - \sum_{i \in \Gamma_n^I(t)} Q_i(t) + Q_n^{ext}(t)}, \quad \forall\, n \in \mathbf{J} \tag{12.1}$$

$$\begin{aligned} V_n(t)\frac{dc_n(t)}{dt} = &\sum_{i \in \Gamma_n^O(t)} Q_i(t)\hat{c}_i^O(t) - \sum_{i \in \Gamma_n^I(t)} Q_i(t)\hat{c}_i^I(t) + m_n(t) \\ &- \left[\sum_{i \in \Gamma_n^O(t)} Q_i(t) - \sum_{i \in \Gamma_n^I(t)} Q_i(t) + Q_n^{\mathrm{ext}}(t) \right] c_n(t), \quad \forall\, n \in \mathbf{ST} \end{aligned} \tag{12.2}$$

$$\frac{\partial \hat{c}_i(x,t)}{dt} + u_i(t)\frac{\partial \hat{c}_i(x,t)}{dx} = 0, \forall i \in \mathbf{P} \tag{12.3}$$

where c_n and m_n denotes the concentration and mass injected at a node, respectively. The variable $\hat{c}_i$ is the concentration inside pipe i and V_n is the volume of water inside tank n. The variable $\mathbf{J}$ is a set of all junctions, $\mathbf{ST}$ is a set of all storage tanks and $\mathbf{P}$ is a set of all pipes. The variable Q denotes volumetric flow rates that are pre-calculated using EPANET 2.00.12 and are assumed to be constant over each hydraulic time step. The flow rate of a known external source entering a node is also pre-calculated and is denoted by Q_i^{ext}. The variable Γ_n^O represents the set of all pipes with flow going away from node n. Similarly, Γ_n^I represents the set of all pipes with flow coming into node n.

Equation 12.1 represents a set of algebraic equations dependent on time alone and Equation 12.2 represents a set of ordinary differential equations (ODEs) also dependent on time alone. Therefore, these two equations can be discretized in time. However, discretizing Equation 12.3, which are partial differential equations (PDEs), in both time and space would lead to a prohibitively large model. Instead, these pipe balance PDEs are replaced using an origin-tracking algorithm. This algorithm is based on the Lagrangian method; however, instead of tracking concentration values as packets of water moving through the network, the origin-tracking algorithm tracks the originating node and time step of each packet as it enters a pipe (see Figure 12.1). Once the water packet exits the pipe, equations are written relating the concentration of the pipe inlet and outlet to the concentration of connected nodes based on time delay. These time delay expressions are formulated for each pipe independently. Therefore, the algorithm scales favorably for a large water distribution system having a linear computational cost as the size of the network increases.

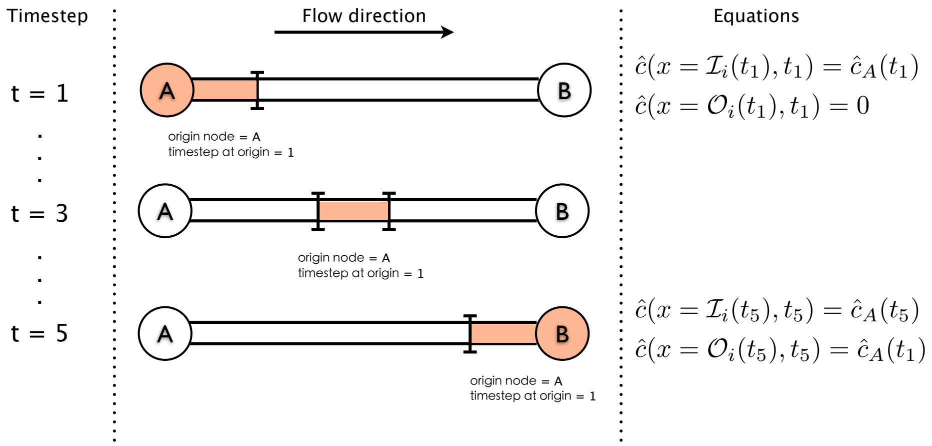

Timestep
Flow direction
Equations
t = 1
A
B
origin node = A
timestep at origin = 1
$\hat{c}(x = \mathcal{I}_i(t_1), t_1) = \hat{c}_A(t_1)$
$\hat{c}(x = \mathcal{O}_i(t_1), t_1) = 0$
t = 3
A
B
origin node = A
timestep at origin = 1
t = 5
A
B
origin node = A
timestep at origin = 1
$\hat{c}(x = \mathcal{I}_i(t_5), t_5) = \hat{c}_A(t_5)$
$\hat{c}(x = \mathcal{O}_i(t_5), t_5) = \hat{c}_A(t_1)$

Figure 12.1: Illustration of the origin tracking algorithm.

By calculating time delay expressions, a very large but sparse linear system of equations is generated that relates input injections (m) from all nodes and time steps to output concentrations (c) from all nodes and time steps.

$$Gc = Dm \tag{12.4}$$

Unlike black box simulations, this linear model can be extended and embedded inside other numerical applications. For example, the water quality model can be embedded inside a mathematical programming formulation for applications like booster placement, source inversion and optimal grab sampling.

After formulating the linear system, performing a tracing simulation is straightforward. First, an injection profile (m) is specified. Then, the system is factorized and finally backsolved for the network concentration profile c. This process is fast, and even more efficient when simulating a large ensemble of tracing simulations. In this case, the system is factorized once, and a backsolve is performed for each simulation. To get additional speedup, a tailored solver is also provided that takes advantage of the structure of the linear system by permuting matrix G into lower triangular, which removes the need for any factorization. The tailored solver also utilizes the Basic Linear Algebra Sub-routines (BLAS) library to perform multiple backsolves (corresponding to multiple injection scenarios) more efficiently. For additional information about Merlion, refer to Mann et al. (2012a).

12.2 Average-case Sensor Placement

The sp subcommmand can design a sensor network for contamination warning systems (CWSs) using a variety of different optimization formulations. The most widely studied sensor placement formulation for CWS design is to minimize the expected impact of an ensemble of contamination incidents given a sensor budget. This formulation has also become the standard formulation for sp, because it can be effectively used to select sensor placements in large water distribution networks. This chapter provides a variety of examples that illustrate the application of the sp subcommand for this optimization formulation. Sensor placement formulation and examples illustrating common use of sp are included in Chapter 5.

12.2.1 Computing a Bound on the Best Sensor Placement Value

A mixed-integer program (MIP) solver like GLPK provide upper and lower bounds on the value of the final solution. For large water distribution systems, it might be prohibitively expensive to perform optimization with a MIP solver. However, computing a lower bound with these solvers might be practical even for large water distribution systems.

The configuration file shown in Figure 12.2 defines a sensor placement problem with the compute bound option set to true in the problem block. This option indicates that the goal for the optimizer is to compute a lower bound on the globally optimal solution, rather than finding a sensor placement. All other options are those previously defined in example 3 of the Sensor Placement Examples (See Section 5.4).

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute bound: True
  compute greedy ranking: True
solver:
  type: glpk
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: sp_bound/Net3
  debug: 0
```

Figure 12.2: The sp configuration file using the GLPK solver to compute a lower bound.

The YAML output file in Figure 12.3 contains the lower bound value. This is the same value as the solution generated by the GRASP heuristic in example 3 of the Sensor Placement Examples (See Section 5.4). In this manner, a MIP solver can be used to evaluate whether a heuristic sensor placement is near-optimal.

The Lagrangian heuristic leverages the structure of the eSP model (Equations 5.1) to guide its search. Specifically, this heuristic computes the optimal values for the integer relaxation of eSP and then applies a ran-

```
# sp output
general:
  version: 1.5                                                  # WST version
  date: '2019-03-05'                                            # Run date
  cpu time: 4.364                                               # CPU time (sec)
  directory: C:/wst-1.5/examples/sp_bound
  log file: Net3sp_output.log                                   # Log file
sensor placement:
  nodes: []                                                     # List of sensor nodes
  objective: null                                               # Objective value
  lower bound: 8655.806356                                      # Lower bound
  upper bound: null                                             # Upper bound
  greedy ranking: Net3_evalsensor.out                           # Upper bound
  stage 2: []                                                   # Upper bound
```

Figure 12.3: The sp YAML file with the lower bound from the GLPK solver.

domized rounding technique. As a consequence, this heuristic can also be used to compute bounds on the value of sensor placement in a manner that is similar to a MIP solver. The configuration file in Figure 12.4 uses the Lagrangian solver to determine the sensor placement for example 3 of the Sensor Placement Examples (See Section 5.4).

```
impact data:
- name: impact1
  impact file: Net3/Net3_ec.impact
  nodemap file: Net3/Net3.nodemap
objective:
- name: obj1
  goal: impact1
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute bound: False
  compute greedy ranking: False
solver:
  type: lagrangian
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: sp_bound_lag/Net3
  debug: 0
```

Figure 12.4: The sp configuration file using the Lagrangian solver.

The YAML output file in Figure 12.5 shows the results of sensor placement using the Lagrangian solver for example 3 of the Sensor Placement Examples (See Section 5.4). It contains the sensor locations (EPANET IDs), the objective value (the impact metric value), the lower bound on this objective as well as the upper bound, which is the same as the objective value. The sensor locations identified are nodes 139, 161, 191 and 208. The mean extent of contamination (EC) impact for this design is approximately 9889 pipe feet contaminated. The lower bound is approximately 9819 pipe feet contaminated, which is greater than the bound computed by GLPK. This illustrates the fact that the bounds computed by the Lagrangian solver are weaker than those computed by a MIP solver.

```
# sp output
general:
  version: 1.5                                                       # WST version
  date: '2019-03-05'                                                 # Run date
  cpu time: 0.501                                                    # CPU time (sec)
  directory: C:/wst-/examples/sp_bound_lag
  log file: Net3sp_output.log                                        # Log file
sensor placement:
  nodes: [['139', '161', '191', '208']]                              # List of sensor nodes
  objective: ['9889.90378']                                          # Objective value
  lower bound: 9819.335391                                           # Lower bound
  upper bound: 9889.90378                                            # Upper bound
  greedy ranking: Net3_evalsensor.out                                # Upper bound
  stage 2: []                                                        # Upper bound
```

Figure 12.5: The sp YAML file with the lower bound from the Lagrangian solver.

As with MIP solvers, the Lagrangian solver can also be used to simply compute this lower bound. The configuration file in Figure 12.6 shows an example of using the compute bound option in the problem block with the Lagrangian solver.

12.2.2 Managing Sensor Placement Locations

By default, the sp subcommand assumes that all node locations in a water distribution network are feasible sensor locations. In practice, sensors cannot be practically installed in many locations without significant cost and inconvenience. The location block in the configuration file is used to specify options for declaring feasible and infeasible node locations in the network. Additionally, the location block can be used to declare node locations as fixed, where a sensor must be placed, and unfixed, where a sensor cannot be located.

A location block consists of a list of declarations that are interpreted in their order within the configuration file. Each declaration consists of a dictionary with a single key, whose value is either a string or list of EPANET node IDs. For example, the following location block declares a list of infeasible node locations:

```
  location:
    - infeasible nodes:
      - 113
      - 121
      - 141
      - 163
      - 209
```

The impact of infeasible sensor locations on the results for example 1 of the Sensor Placement Examples (See Section 5.4) is shown in following example. The solution from this example placed sensors at nodes 113, 121, 141, 163 and 209 and the mean extent of contamination (EC) for this sensor design was 8655. If these nodes were listed as infeasible sensor locations (as shown in the location block above) in the configuration file, the new sensor locations are nodes 111, 119, 169, 207 and 237. The mean EC for this new solution is 8932 which is worse than the initial design; this reflects the fact that a sensor design that can use any location will be better than a sensor design that can use a limited set of locations.

12.2.3 Limited-Memory Sensor Placement Techniques

Controlling the memory used by optimizers is a critical issue when solving large sensor placement formulations. This is a particular challenge for MIP methods, but both the GRASP and Lagrangian heuristics can exceed a workstation's memory when solving very large problems. The sp subcommand supports a variety of mechanisms that reduce the problem representation size while preserving the structure of the sensor

```
impact data:
- name: impact1
  impact file: Net3_ec.impact
  nodemap file: Net3.nodemap
  directory: Net3
objective:
- name: obj1
  goal: impact1
  statistic: MEAN
constraint:
- name: const1
  goal: NS
  statistic: TOTAL
  bound: 5.0
sensor placement:
  type: default
  objective: obj1
  constraint: const1
  presolve: True
  compute bound: True
  compute greedy ranking: False
solver:
  type: lagrangian
  options: {}
  logfile: null
  verbose: 0
configure:
  output prefix: sp_bound_only_lag/Net3
  debug: 0
```

Figure 12.6: The `sp` configuration file using the Lagrangian solver and the compute bound option.

placement problem. These techniques include: scenario aggregation and filtering, feasible locations, witness aggregation, skeletonization and explicit memory management.

Scenario Aggregation: Scenario aggregation compresses the data in an impact file while preserving its fundamental structure. This strategy is effective when optimizing for mean performance objectives. Scenario aggregation is performed with the `scenarioAggr` command, which is described in Section 14.4.

Filtering Impacts: Filtering impacts can also reduce memory requirements for sensor placement by reducing the size of the impact files. Filtering can limit the sensor placement formulation to only consider contamination incidents that are sufficiently bad in the worst-case. Filtering is performed with the `filter_impacts` executable, which is described in Section 14.2

Feasible Locations: Limiting the feasible locations is another strategy to reduce memory requirements. The size of the sensor placement formulation decreases as the number of feasible locations decreases. The location block option described in Section 12.2.2 can be used to specify the set of feasible locations.

Witness Aggregation: Witness aggregation limits the size of the sensor placement formulation by aggregating the decision variables that witness a contamination incident. By default, variables that witness contamination incidents with the same impact are aggregated, and this typically reduces the MIP constraint matrix by a significant amount. Further reductions perform more aggressive aggregations that create an approximate sensor placement formulation.

Witness aggregation is specified using an `aggregate` block in the `sp` configuration file. A named aggregation block specifies the type of aggregation, the aggregation limit value and the associated impact data. For example:

```
aggregate:
- name: agg1
  type: PERCENT
  goal: impact1
  value: 0.125
  conserve memory: 0
  distinguish detection: 0
  disable aggregation: [0]
```

The following table illustrates the use of the two witness aggregation options when optimizing the mean extent of contamination: aggregation type = PERCENT and aggregation type = RATIO. The RATIO aggregation type can be used with distinguish detection option to help with aggregation. The second line of data in this table is the default aggregation, which has about half as many non-zero values in the MIP constraint matrix. Both the percent and ratio aggregation strategies effectively reduce the problem size while finding near-optimal solutions.

Aggregation Type	Aggregation Value	Binary Variables	MIP Nonzeros	Solution Value
None	NA	97	220736	8525
PERCENT	0.0	97	119607	8525
PERCENT	0.125	97	49576	9513
RATIO	0.125	97	12437	10991

Skeletonization: Another option to reduce the memory requirement for sensor placement is to reduce the size of the network model through skeletonization. Skeletonization groups neighboring nodes based on the topology of the network and pipe attributes. Section 14.5 describes the `spotSkeleton` executable, which provides techniques for branch trimming, series pipe merging and parallel pipe merging. These techniques eliminate pipes and nodes that have little effect on the overall hydraulics of the system. This effectively contracts a connected piece of the network into a single node, called a supernode. Skeletonized networks can be used to define geographic proximity in a two-tiered sensor placement approach for large network models (Klise et al., 2013).

Explicit Memory Management: The GRASP heuristic has several options for controlling how memory is managed. The `grasp-representation` solver option can be used to control how the local search steps are performed. By default, a dense matrix is precomputed to perform local search steps quickly, but a sparse matrix can be used to perform local search with less memory. Also, the GRASP heuristic can be configured to use the local disk to store this matrix.

12.2.4 Evaluating a Sensor Placement

Sensor placements can be evaluated based on an impact assessment of possible contaminant incidents. The `evalsensor` executable measures the performance of each sensor placement with respect to the set of possible contamination locations. This analysis assumes that probabilities have been assigned to these contamination locations. If no probabilities are given, then uniform probabilities are used. The `evalsensor` executable takes sensor placements in a sensor placement file and evaluates them using data from one or more impact files. Sensor placement files are generated using the `sp` subcommand, and the file format is described in File Formats Section 13.10. Impact files are generated using the `sim2Impact` subcommand, and the file format is described in the File Formats Section 13.4. Additional information on `evalsensor` can be found in the Executable Files Section 14.1.

The following example demonstrates the use of `evalsensor` using the sensor network design from Section 5.4. The `evalsensor` command for this example is executed using the following command:

```
evalsensor --nodemap=Net3.nodemap Net3_ec.sensors Net3_ec.impact Net3_mc.impact
```

This example generates output shown in Figure 12.7.

```
-------------------------------------------------------------------------------
Sensor placement id:        23112
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            17 19 24 65 88
Sensor junctions:           115 119 127 209 267

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                9951.5669
Lower quartile impact:      1650.0000
Median impact:              9694.0000
Upper quartile impact:      15044.8000
Value at Risk (VaR) (  5%): 25485.0000
TCE                 (  5%): 27992.4667
Max impact:                 33110.0000

Impact File:                Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                70806.1310
Lower quartile impact:      503.9170
Median impact:              83999.3000
Upper quartile impact:      143984.0000
Value at Risk (VaR) (  5%): 143999.0000
TCE                 (  5%): 144049.5000
Max impact:                 144143.0000
-------------------------------------------------------------------------------
```

Figure 12.7: The `evalsensor` example output.

The `evalsensors` command can also evaluate a sensor placement in the case where sensors can fail, and there is some small number of different classes of sensors (grouped by false negative probability). This information is specified by an imperfect sensor class file and an imperfect junction class file, which are defined in Sections 13.6 and 13.5, respectively. The imperfect sensor class (sc) file, `Net3.imperfectsc`, specifies different categories of sensor failures. Sensors of class 1 have a false negative probability of 25%, sensors of class 2 have a probability of 50%, class 3 have a 75% probability and class 4 100%.

```
1 0.25
2 0.50
3 0.75
4 1.0
```

Once failure classes are defined, the nodes of the network are assigned to failure classes by using a imperfect junction class (jc) file. The beginning of the imperfect junction class file Net3.imperfectjc is shown below.

```
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
```

Given the junction classes, `evalsensor` is used to determine the expected impact of a sensor placement, given that sensors might fail. The following command line executes `evalsensor` with specified failure probabilities:

```
evalsensor --nodemap=Net3.nodemap --sc-probabilities=Net3.imperfectsc \
    --scs=Net3.imperfectjc Net3_ec.sensors Net3_ec.impact
```

This example generates output shown in Figure 12.8.

```
------------------------------------------------------------------------------
Sensor placement id:        23112
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            17 19 24 65 88
Sensor junctions:           115 119 127 209 267

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                16472.1977
Lower quartile impact:      5270.0000
Median impact:              13440.0000
Upper quartile impact:      24199.0000
Value at Risk (VaR) (  5%): 49232.5172
TCE                 (  5%): 52421.7876
Max impact:                 55814.4578
------------------------------------------------------------------------------
```

Figure 12.8: The `evalsensor` output using sensor failure probabilities.

The mean extent of contamination impact changes dramatically if sensors are allowed to fail. The original solution was misleading if sensors fail according to the assigned probabilities. With sensor failures, the expected impact is much higher.

12.3 Source Identification with Grab Samples Case Study

The following case study illustrates how the `inversion` and `grabsample` subcommands can be used in tandem to perform multiple cycles of source inversion calculations as more and more measurement data becomes available. The solution approach integrates iterative sampling strategy for finding the contamination source using discrete measurements obtained from manual grab samples taken during different sampling cycles. Figure 12.9 illustrates the source inversion and grab sample strategy. A contamination incident is first suspected given a customer inquiry or detection from a fixed continuous sensor in the Contamination Warning System (CWS). At this stage, a team is sent out to gather manual grab samples at and around the location of first detection. Discrete yes/no measurements from these manual grab samples along with the measurement from CWS are then used to identify the potential sources of the contamination incident. Since the inversion problem is an ill-posed problem, the solution will generally be non-unique. A set of likely locations can be identified and the `grabsample` subcommand can be used to determine the location of the next manual grab samples. The source inversion is performed again using the new information. This cycle of collecting manual grab samples and performing source inversion is continued until the true injection location(s) is identified.

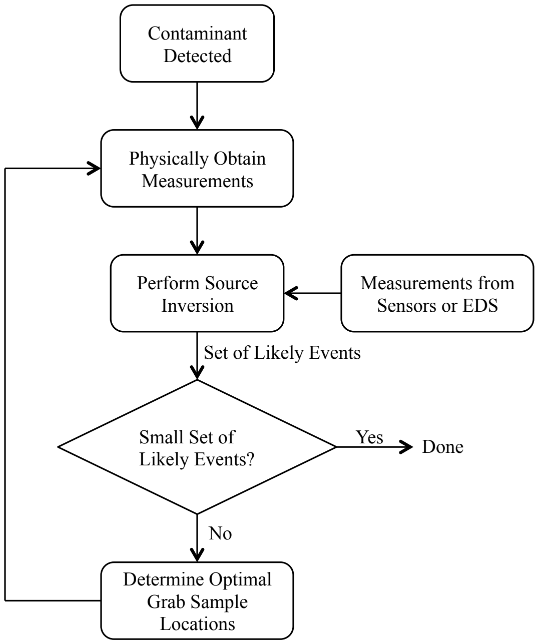

Contaminant Detected
Physically Obtain Measurements
Perform Source Inversion
Measurements from Sensors or EDS
Set of Likely Events
Small Set of Likely Events?
Yes
Done
No
Determine Optimal Grab Sample Locations

Figure 12.9: Illustration of the source inversion and grab sample cycling strategy.

12.3.1 Case Study

Since real system data is not available, the `measuregen` executable is used to generate simulated data for the contamination incident in the following case study. In this simulation, a conservative contaminant is injected into node 163 of EPANET Example Network 3 (Net3) starting at 8 AM. The Bayesian probability based formulation [8.1.2] is used in the `inversion` subcommand to identify the possible contamination sources. Figure 12.10 shows the fixed sensor locations in blue while the original contamination location is shown in red. The CWS consists of five fixed sensors that provide measurements every 15 minutes (set as a command line option in `measuregen`).

All the files required for this case study are provided in the `examples/case_studies/inversion` folder.

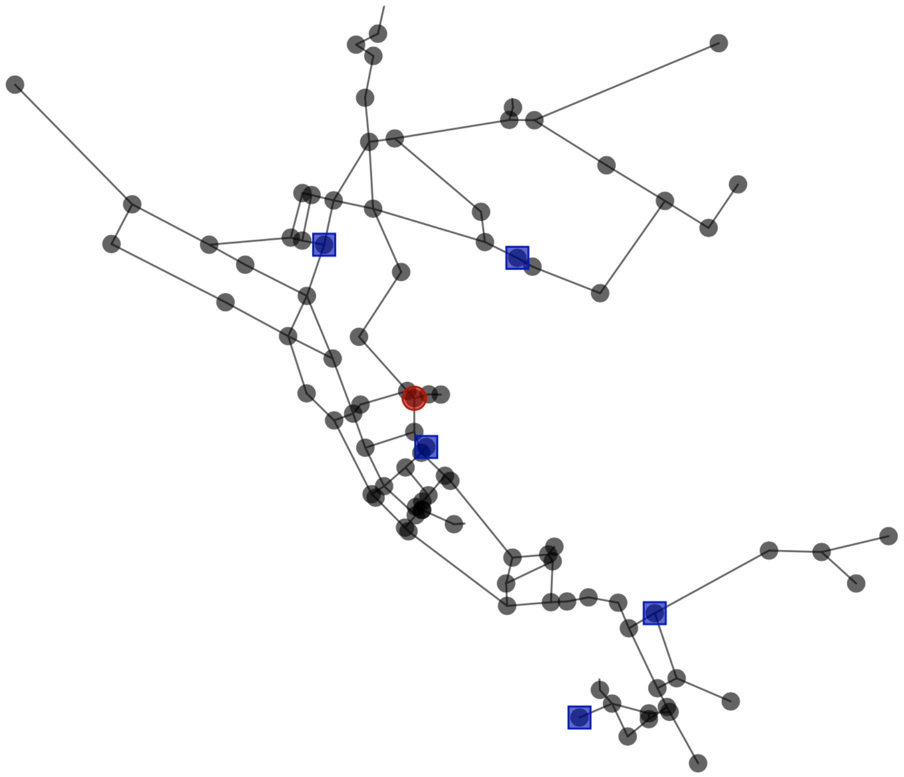


Figure 12.10: Fixed sensors (blue) and contamination location (red) for case study.

The case study is a composed of three cycles of source inversion and grab sampling to reduce the number of possible contamination source locations. During each cycle, the `inversion` subcommand uses the following data:

- Net3.inp - EPANET Example Network 3 input (INP) file.
- MEASURES.dat - Measurements file binary (yes/no) results from fixed sensors and grab samples (generated using `measuregen` executable).
- <*output_prefix*>_Likely_Nodes.dat - Likely nodes file containing a list of feasible nodes to consider as possible contamination source nodes. This file is only used in cycles 2 & 3.

The `inversion` subcommand outputs a YAML file with a list of possible contamination sources. A TSG file is also created that provides information about the possible contamination incidents. This file can be used in the `grabsample` subcommand. Thereafter, Cycles 1 & 2 use the following data to determine optimal sample location:

- Net3.inp - EPANET Example Network 3 input (INP) file.
- <*output_prefix*>_profile.tsg - TSG file which contains a list of likely injections obtained from the `inversion` subcommand from the preceding cycle.
- Sample time - Time in the future when the samples are expected to be taken. This is generally the current time plus the time it would take the sample teams to obtain measurements.

- fixed_sensors.dat - List of fixed continuous sensor locations. This is used to avoid fixed sensors being selected as grab sample locations.

12.3.2 Cycle 1

At 8:15 AM, the sensor located at node 167 detects abnormal water quality. Further confirmation is made with another positive measurement at 8:30 AM. At this point, the measurement data from the past 8 hours is used to perform source inversion. The `inversion` subcommand identifies 24 possible injection locations as shown in red in Figure 12.16a. The `grabsample` subcommand is then used to identify additional measurement locations that will reduce the number of possible injection locations. The utility has three teams available to gather manual grab samples and it takes 30 minutes for each team to obtain the manual samples. The `grabsample` subcommand identifies the three optimal grab sample locations shown in blue and the possible injection nodes in red in Figure 12.16a.

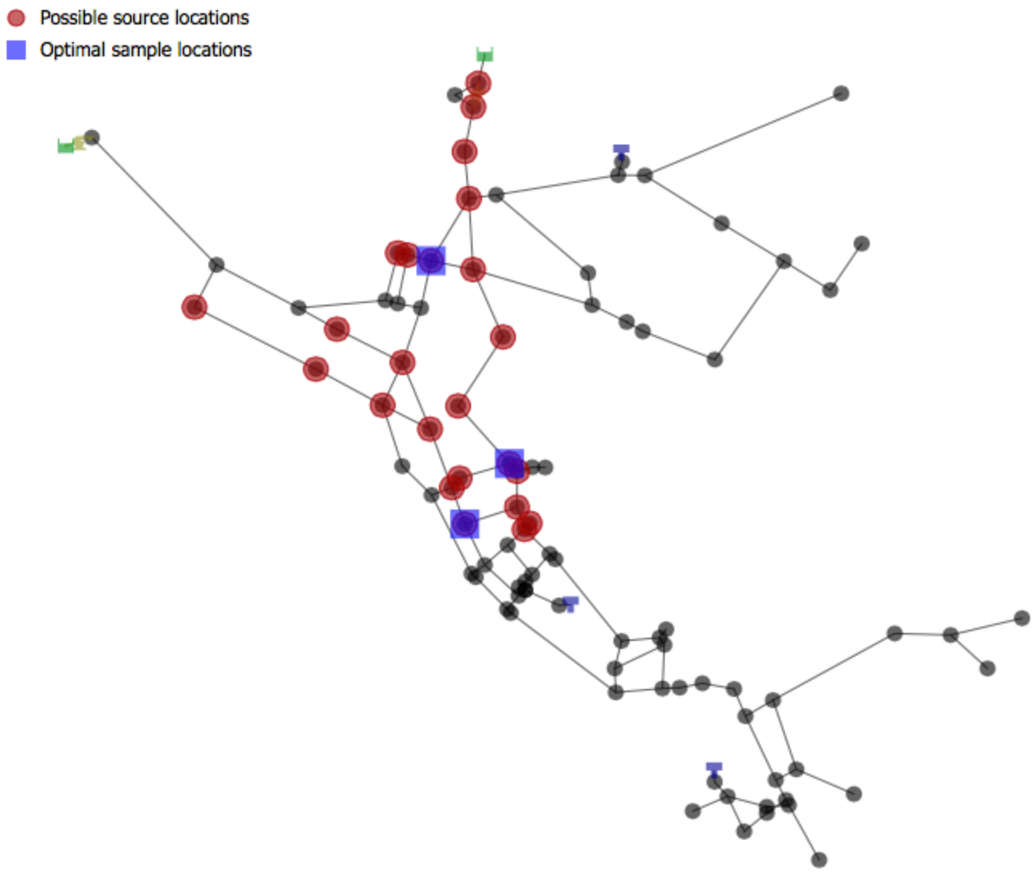


Figure 12.11: Cycle 1 identified optimal grab sample locations (blue).

The files required and generated during this cycle of source inversion and grab sample are provided in the `examples/case_studies/inversion/Cycle1` folder.

12.3.3 Cycle 2

In the 30 minutes that the sampling teams take to get manual grab sample measurements from the locations identified in Cycle 1, new measurements are also available from the fixed sensors in the CWS. It is assumed that the sampling teams have field instruments that can provide them with a yes/no indication of the presence or absence of a contaminant. At 9:00 AM, these new measurements are used to perform source inversion again. This time the `inversion` subcommand identifies seven possible injection locations as shown in Figure 12.16b. In Cycle 2, the possible sources were restricted to the 24 nodes identified in Cycle 1 using the feasible nodes option. Again a 30-minute delay for sample collection and three sample teams were used in the `grabsample` subcommand to identify the optimal grab sample locations at 9:30 AM. These grab sample locations (blue) and the possible injection nodes (red) are shown in Figure 12.16b.

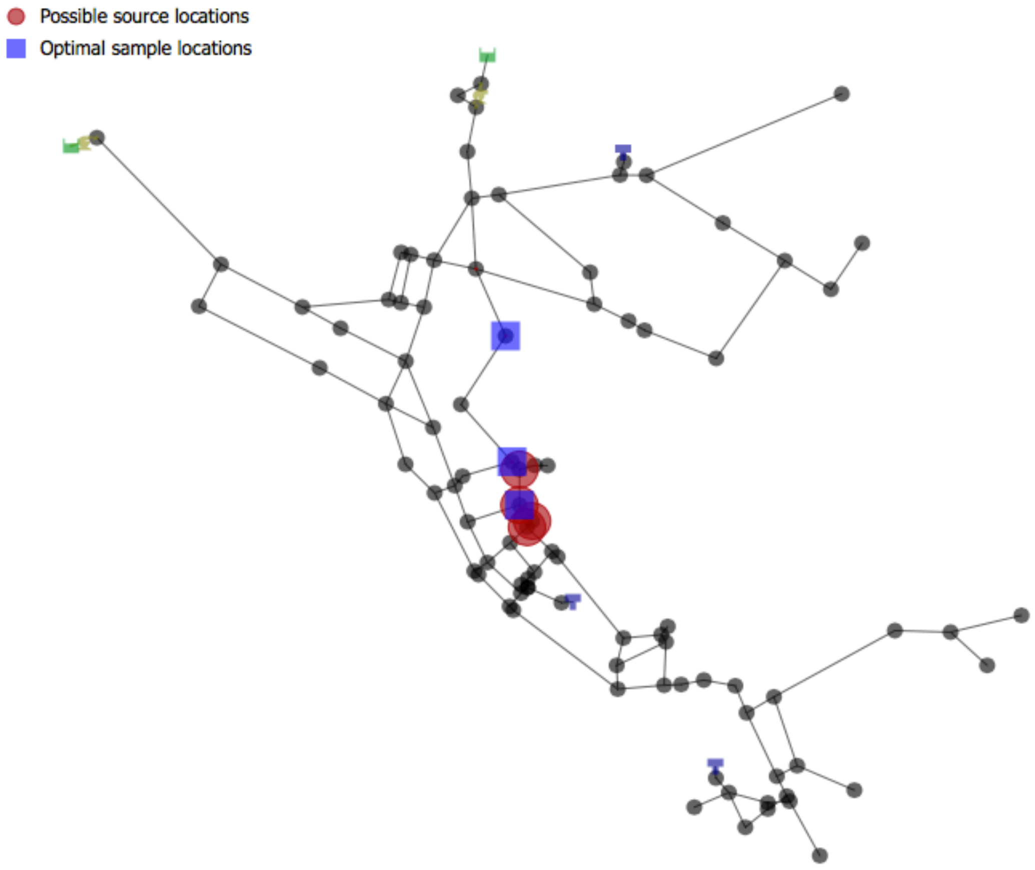


Figure 12.12: Cycle 2 identified optimal grab sample locations (blue).

The files required and generated during this cycle of source inversion and grab sample are provided in the `examples/case_studies/inversion/Cycle2` folder.

12.3.4 Cycle 3

Grab sample measurements are obtained at 9:30 AM from the optimal locations identified in Cycle 2. These along with the new measurements obtained from the fixed sensors are used to perform source inversion again. Only the seven possible injection nodes obtained in Cycle 2 are considered as feasible nodes in the `inversion` subcommand for Cycle 3. Three possible injection locations as shown in Figure 12.16c are identified in this cycle.

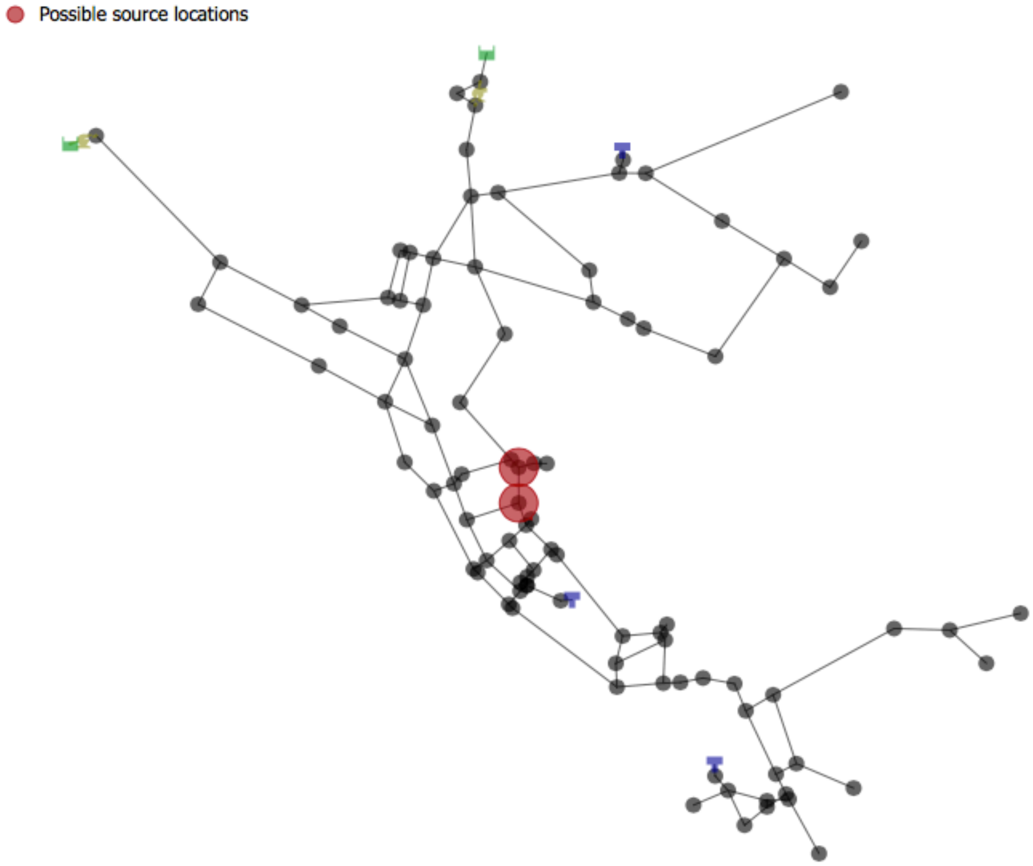


Figure 12.13: The possible injection nodes (red) identified in Cycle 3.

Since the water utility has three sampling teams available, the teams can directly inspect the three possible injection locations identified in this cycle to confirm the true injection location.

12.4 Uncertainty Reduction with Grab Samples Case Study

The following case study illustrates how the functionality of the `inversion`, `grabsample` and `uq` subcommands can be used in tandem to identify multiple sampling cycles that reduce uncertainty in the extent of contamination. The approach uses successive measurements obtained from manual grab samples to find the contamination plume. Figure 12.14 illustrates the methodology. A contamination incident is first suspected following a customer inquiry or detection from a fixed continuous sensor in the Contamination Warning System (CWS). At this stage, a team is sent out to gather manual grab samples at and around the location of first detection. Discrete yes/no measurements from these manual grab samples along with the measurement from contamination warning system are then used to estimate the probability that nodes in the network are contaminated. The probability of node contamination provides a metric of uncertainty quantification. Given a particular confidence level (e.g., 95%), nodes can be categorized according to their probability of contamination: LY for "likely yes," LN for "likely no" and UN for "unknown."

If a sufficiently small number of nodes remain uncertain, then the process is terminated, otherwise further sampling cycles are required. This cycle of collecting manual grab samples is continued until the contamination plume is estimated with a good level of confidence.

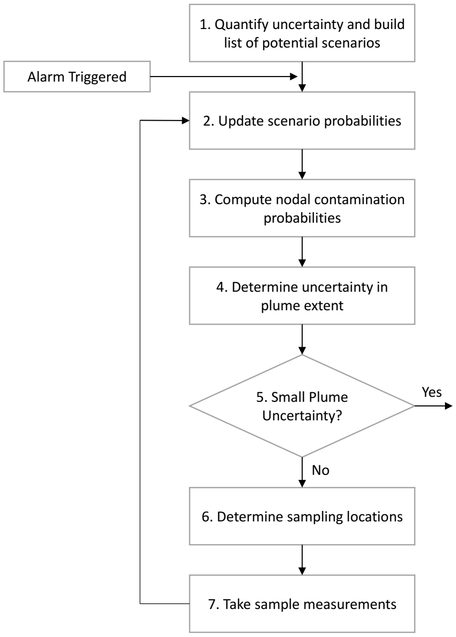

1. Quantify uncertainty and build list of potential scenarios
Alarm Triggered
2. Update scenario probabilities
3. Compute nodal contamination probabilities
4. Determine uncertainty in plume extent
5. Small Plume Uncertainty?
Yes
No
6. Determine sampling locations
7. Take sample measurements

Figure 12.14: Illustration of the source inversion and grab sample cycling strategy.

12.4.1 Case Study

Since real system data is not available, the `measuregen` executable is used again to generate simulated data for a contamination incident in the following case study. In this simulation, a conservative contaminant is injected into node 115 of EPANET Example Network 3 (Net3) at 8 am. The set of potential scenarios includes 10 hydraulic realizations with 97 different injection locations. All of the files required for this case study are provided in the `examples/case_studies/sampling` folder.

The case study is composed of four sampling cycles to reduce uncertainty in the extent of contamination. The initial warning is raised at 10 am. From this time, the procedure in Figure 12.14 is followed to reduce uncertainty by gathering grab samples every hour. In each sampling cycle, three additional samples are collected. Given the information provided by the new samples, the probability distribution of scenarios is updated following Bayesian statistics. Then, the methodology proposed in Section 9.1 is followed to quantify uncertainty. To facilitate the analysis, a script was implemented in Python to run the sampling cycles in a loop, which is the signals.py file provided in the `examples/case_studies/sampling/cycling` folder. The execution of the script follows the same convention as the WST subcommands:

```
python wst/packages/pywst/pywst/cycling/signals.py cycle <configfile>
```

The options for the script are the same as the options provided in the sections of the `inversion`, `uq` and `grabsample` subcommands. Some additional options to specify duration of each cycle and number of cycles are added. The configuration file for this case study, sampling_case_study.yml, is shown in Figure 12.15.

```
confidence nodes: 0.90
confidence scenarios: 0.99
delta t: 60
measurements: MEAS.dat
num cycles: 6
mip solver: glpk
number of samples: 3
set scenarios: list_scenarios
start time: 120
threshold: 0.01
true scenario: list_true_scenario
sample criteria: probability1
prune scenarios: True
```

Figure 12.15: The configuration file for sampling case study.

12.4.2 Cycle 0

At 10 AM, the contamination warning system detects abnormal water quality at nodes 40 and 111. The measurement data from those two locations is used to perform a Bayesian update in the probability of scenarios. At this point, given the probability distribution of scenarios, 73 possible scenarios are identified as most likely. The uncertainty in the number of scenarios is evident in the uncertainty quantification as most of the nodes are deemed unknown (UN). Figure 12.16a shows in yellow all locations that are considered uncertain to have contamination. The utility has three teams available to gather manual grab samples and it takes 60 minutes for each team to obtain the manual samples. The probability-based formulation in Chapter 10 identifies the three optimal grab sample locations shown in black in Figure 12.16a.

12.4.3 Cycle 1

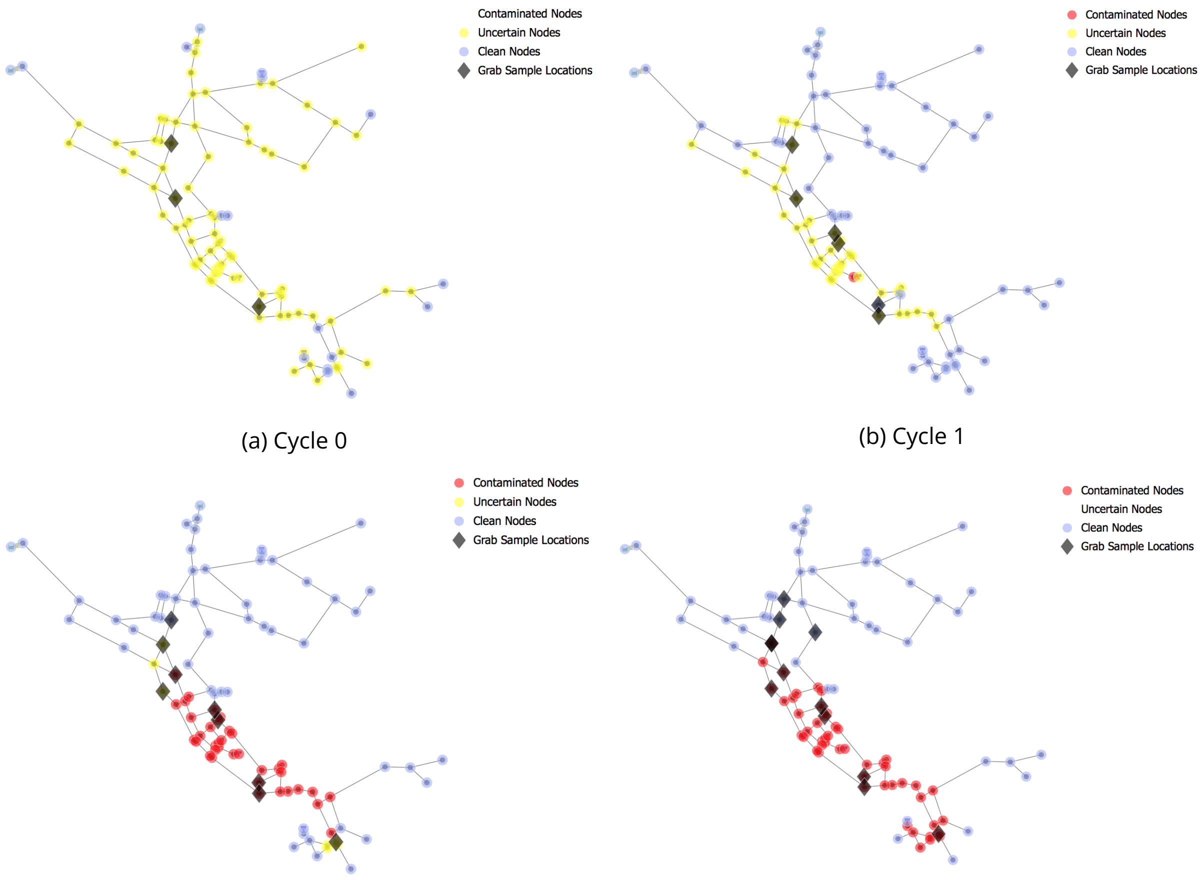
At 11:00 AM, the new measurements are used to perform a Bayesian update and an uncertainty quantification. This time the number of uncertain nodes is reduced by half as shown in Figure 12.16b. Nodes in red are likely to be contaminated (LY) and nodes in blue are likely to not be contaminated (LN). Again a 60-minute delay for sample collection and three sample teams were used in the probability-based formulation in Chapter 10 to identify the optimal grab sample locations at 12:00 PM. The three new and three previous grab sample locations (black) are shown in Figure 12.16b.

12.4.4 Cycle 2

Grab sample measurements are obtained at 12:00 PM from the optimal locations identified in second cycle of the `grabsample` subcommand. These are used to perform Bayesian update and an uncertainty quantification again. Only seven nodes remain uncertain (UN) as to whether they are contaminated in this cycle. As the uncertainty is still not small enough, three more sampling locations are identified by solving the probability based formulation of Chapter 10. The three new grab sample locations plus the six previous (black) are shown in Figure 12.16c.

12.4.5 Cycle 3

Grab sample measurements are obtained at 1:00 PM from the optimal locations identified in in third cycle of the `grabsample` subcommand. With this new information the Bayesian update and the uncertainty quantification lead to zero nodes classified as uncertain (all are likely yes or likely no). The three new grab sample locations plus the nine previous (black) are shown in Figure 12.16d. Only 11 grab sample locations are shown in this cycle since one of the locations was selected to be sampled twice, since different sampling times can provide different measurement data.



(a) Cycle 0

(b) Cycle 1

(c) Cycle 2

(d) Cycle 3

Figure 12.16: EPANET Example Network 3 with grab sample locations (black diamonds), contaminated nodes (red circles), uncertain nodes (yellow circles), and clean nodes (grey circles) identified for each of the cycles in the case study.

12.5 Flushing with Source Identification Case Study

When a water utility becomes aware of a water quality issue either through customer complaints or water quality sensor alarms, they often open a hydrant to flush a portion of the distribution network in order to bring new water into the area and increase the chlorine residual. This case study examines how WST can be used to identify effective flushing locations following a water quality sensor alarm using the the `flushing`, `inversion` and `grabsample` subcommands. All files required to run the case study are provided in the `examples/case_studies/flushing` folder.

The EPANET input file for this example is Net6.inp, which has a simulation duration of seven (7) days starting at midnight. The network is assumed to include a contamination warning system (CWS) with ten optimally placed water quality sensors and an event detection system in operation. The 10 sensors are located at JUNCTION-1617, JUNCTION-199, JUNCTION-2297, JUNCTION-2716, JUNCTION-2930, JUNCTION-3023, JUNCTION-435, JUNCTION-552, JUNCTION-675 and JUNCTION-831. The sensors were optimally placed using the `sp` subcommand. Figure 12.17 shows the water distribution network and the location of the water quality sensors. The CWS provides binary values every 15 minutes from each sensor location. The binary value is zero if water quality conditions are normal or one if the conditions are abnormal.

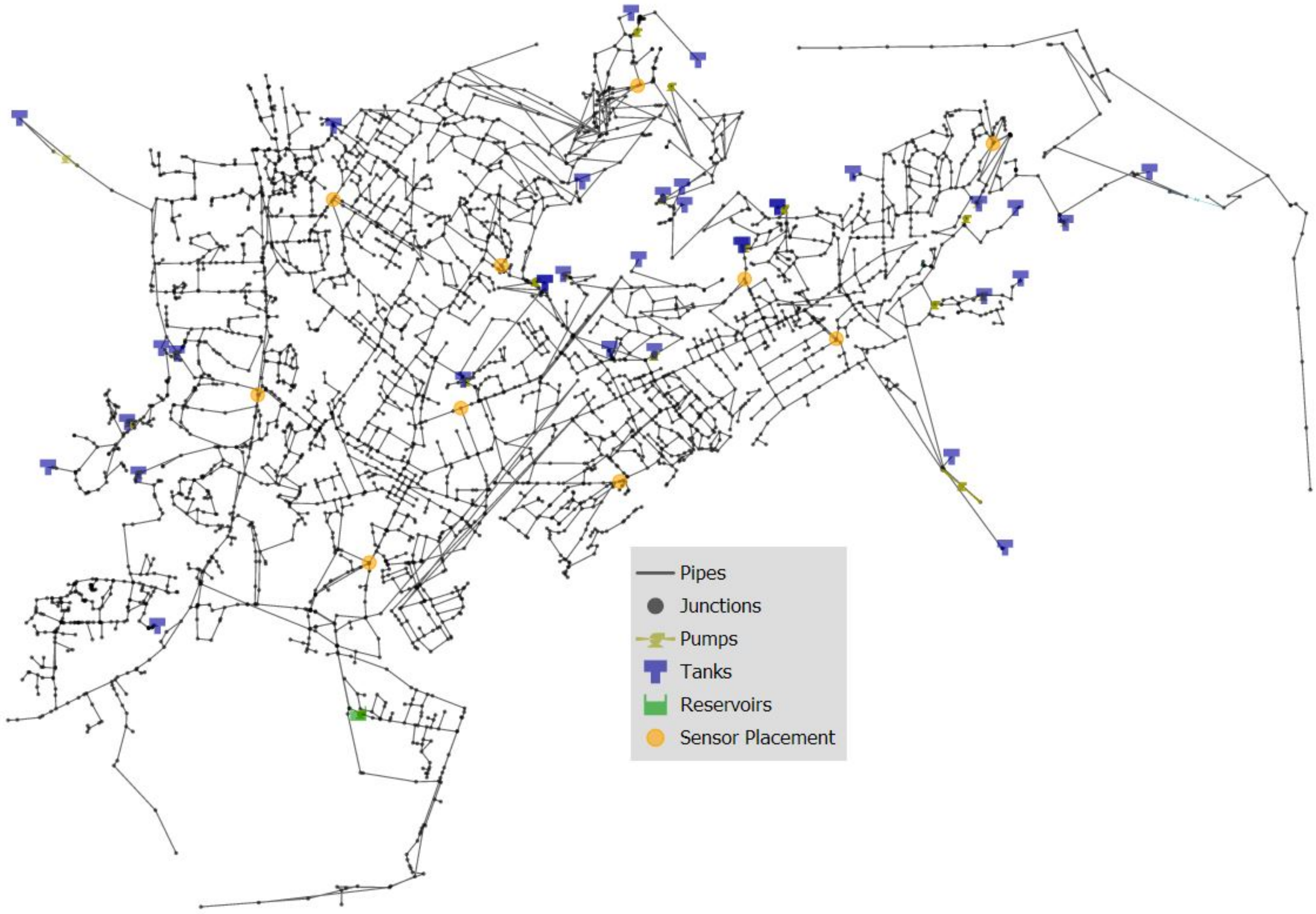


Figure 12.17: Net6 water distribution network with water quality sensors.

At 10:15 AM, the CWS alerts water utility staff to abnormal water quality occurring at water quality sensor located at JUNCTION-1617 in water distribution network model. Figure 12.18 shows the JUNCTION-1617 highlighted as the sensor location with a positive detection of contamination in the network.

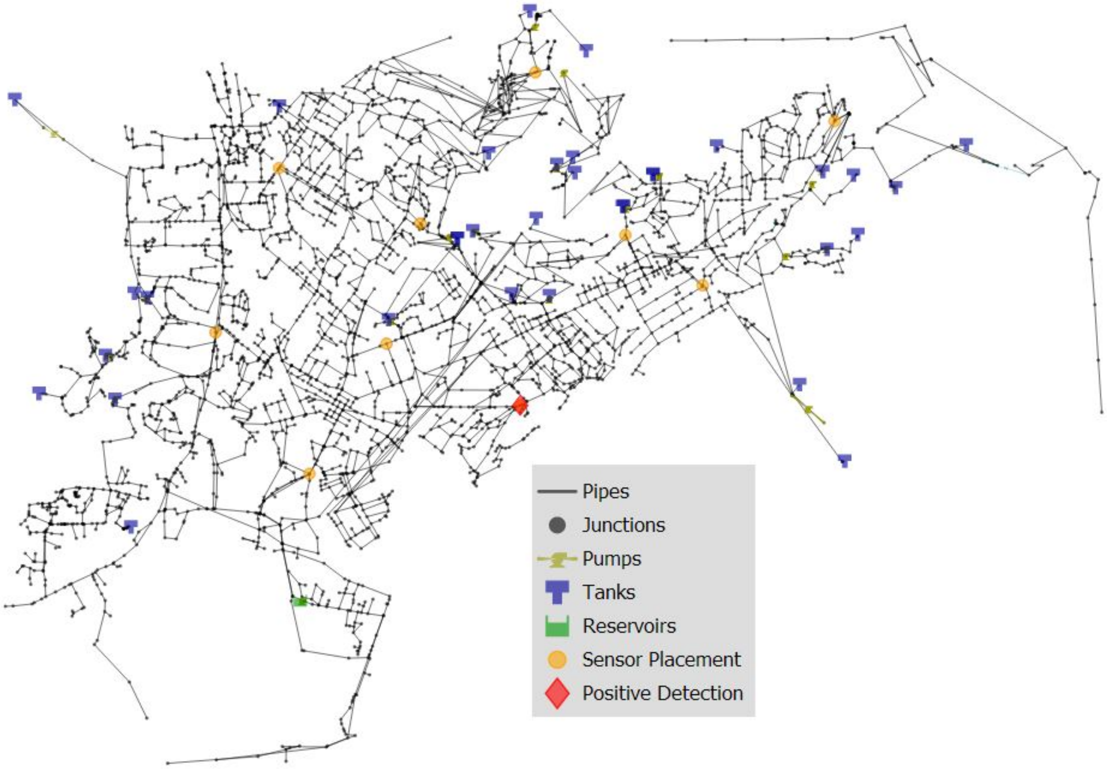


Figure 12.18: Net6 with positive contamination detection at JUNCTION-1617.

The water utility must now decide how to proceed. The staff checks their consequence management plan and sends out a team to ensure that the water quality sensor is working properly. The water utility staff determines that a contamination incident is possible and they would like to identify the source. Source identification allows the water utility to determine the extent of contamination (or spread) and possibly shut off any continuing injection of contaminants. Using the CWS information from the past 35 hours, provided in the measurements file Net6_CWS_MEASURES.dat, and the Net6 INP file, the `inversion` subcommand is used to identify the possible sources of the contamination. The inversion configuration file, Net6_inversion.yml, and the measurements file are provided in the `examples/case_studies/flushing` folder.

The `inversion` subcommand can be executed using the following command line:

```
wst inversion Net6_inversion.yml
```

Figure 12.19 shows the 25 possible contamination sources identified by the `inversion` subcommand.

Since flushing is a common response to abnormal water quality, the water utility staff decide to open hydrants to flush the contaminated water out of the network. To determine the most effective flushing locations, the staff simulates contamination incidents from each possible contamination source location using the TSG file produced from the `inversion` subcommand. From these simulations, the possible extent of contamination from each source location is identified. The nodes in the water distribution network model which are calculated to have contaminant concentrations above zero at the starting of flushing (12:00 PM, approximately two hours after detection) are considered as the initial starting points for the network solver

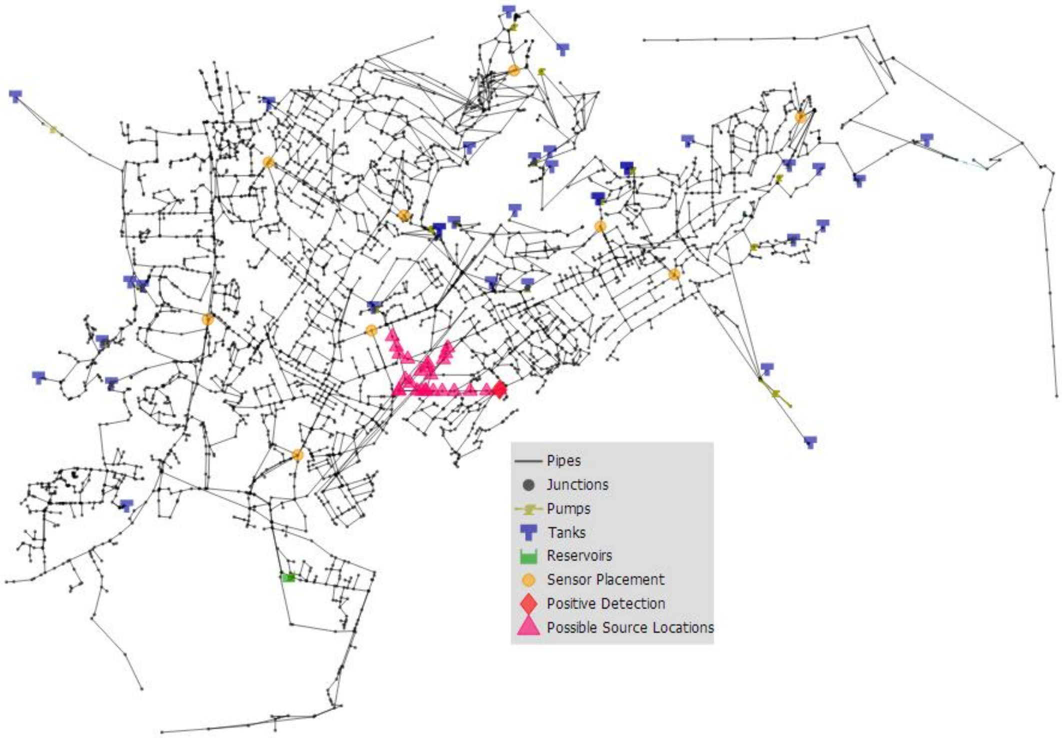


Figure 12.19: Net6 with possible contamination sources identified by `inversion` subcommand.

option in the `flushing` subcommand. These initial starting points are the first node locations that are going to be evaluated in terms of the impact metric and then as the process continues, the solver will look at all of the nodes that are connected to these initial points to determine their impact metrics. Figure 12.20 shows the nodes impacted by the 25 possible contamination source locations.

The water utility decides to open two hydrants to flush the contaminated water out of the network, since the extent of contamination for the possible 25 contamination sources (as seen in Figure 12.20) is not very large at the start of flushing at 12:00 PM. To identify effective flushing locations, the `flushing` subcommand is used. This command requires the following files as specified in the flushing configuration file, Net6_flush_2nodes.yml:

- Net6.inp - Net6 EPANET input (INP) file.
- Net6_inv1_profile.tsg - The TSG file created by the `inversion` subcommand.
- Net6_bio.tai - The TAI file describing the dose-response characteristics for the assumed contaminant. This file is required when using the population exposed (PE) impact metric.

In addition, characteristics of the flushing response are also defined in the flushing configuration file. These include:

- A list of nodes that can be flushed - All non-zero demand (NZD) nodes
- The maximum number of nodes which can be flushed simultaneously - 2
- The flushing rate - 1100 gallons/min

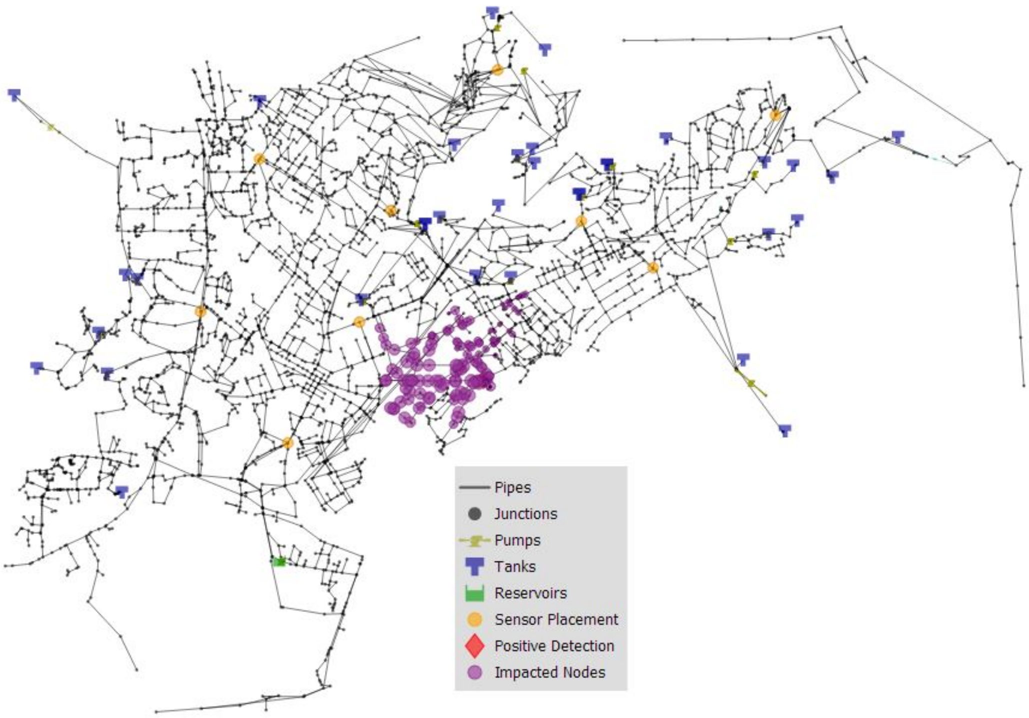


Figure 12.20: Net6 with nodes impacted by the 25 possible contamination sources.

- The flushing duration - 8 hours
- The response time delay (time between detection and start of flushing) - 1 hour

Other information provided in the flushing configuration file include the impact metric that is going to be minimized (PE), the nodes where water quality sensors are located, the type of solver (network solver), and the initial starting points for the network solver (JUNCTION-1881 and JUNCTION-1878).

The `flushing` subcommand can be executed using the following command line:

```
wst flushing Net6_flush_2nodes.yml
```

Figure 12.21 shows the flushing nodes identified by the `flushing` subcommand. The flushing nodes identified are JUNCTION-1881 and JUNCTION-2233.

Since the identified flushing nodes were based upon the 25 possible contamination sources, the water utility staff evaluate the flushing response against each of the possible sources assuming it was the true source of the contamination. This option is available using EVALUATE as the type under the solver block of the flushing configuration file. An example flushing configuration file for the evaluate option is provided in Net6_flush_2nodes_eval_JUNC1617.yml in the `examples/case_studies/flushing` folder. This example assumes that JUNCTION-1617 is the true source of contamination in the network and it evaluates the effectiveness of the identified flushing locations in terms of the PE metric. If JUNCTION-1617 is the true source, flushing at JUNCTION-1881 and JUNCTION-2233 reduces the PE metric by only two percent (2%).

The `flushing` subcommand for this example can be executed using the following command line:

```
wst flushing Net6_flush_2nodes_eval_JUNC1617.yml
```

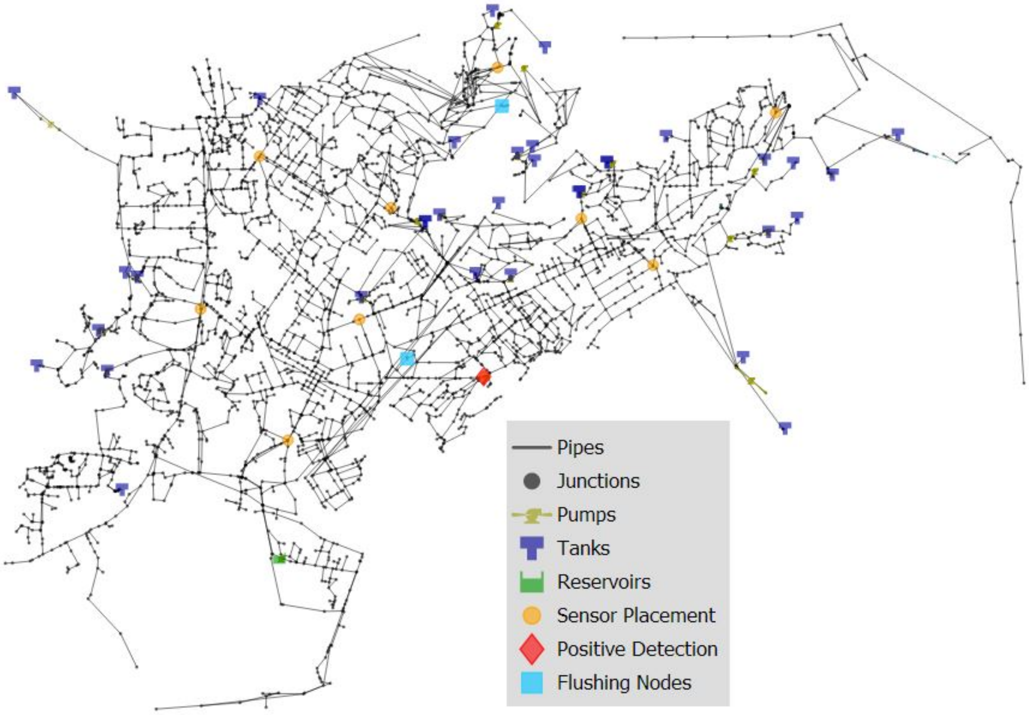


Figure 12.21: Net6 with the flushing nodes identified by the `flushing` subcommand.

Because the `inversion` subcommand solvers assume a continuous injection, the created TSG file has the contamination injection durations lasting as long as the simulation duration listed in the network INP file. Thus, the contamination injections start a little before the detection time and stop at the end of the simulation (seven days). Since an important response action would include shutting off the source of contamination, the TSG file is modified to stop the injection five hours after detection. Using the modified TSG file, the flushing response is evaluated against each of the 25 possible sources assuming it was the true source of the contamination. Figure 12.22 shows the percent reduction in the PE metric for each of the 25 possible contamination sources with flushing alone (blue) and flushing with shutting off the contamination source (green). The percent reduction in the PE metric ranges from 24% to 97% for the flushing with source shut-off response action, which is an increase from the range of 2% to 45% for the flushing alone action. The highest percent reduction was if the true injection incident occurred at JUNCTION-1881.

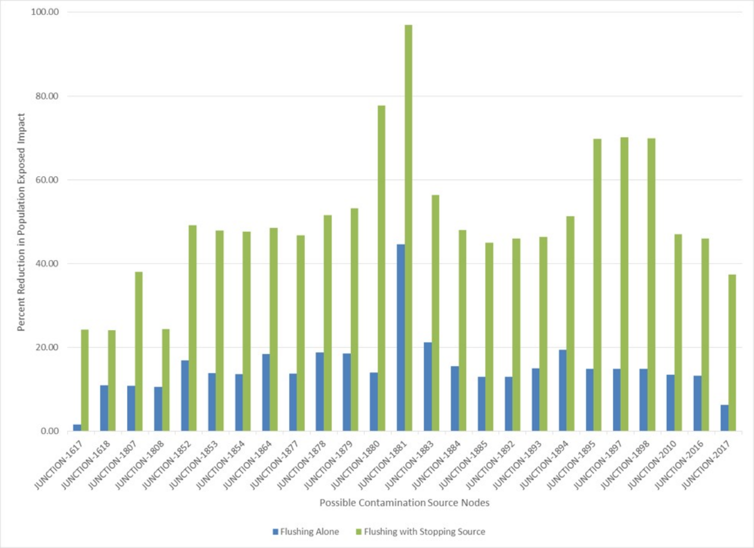


Figure 12.22: The reduction in the PE metric for each of the 25 possible contamination sources.

Chapter 13

File Formats

This chapter describes the different file formats used by WST, including a brief description, format, the associated subcommand(s) and any additional details.

13.1 Configuration File

- **Description:** Input configuration file for all WST subcommands.
- **Format:** YAML
- **Created by:** Template input configuration files can be created using the `--template` option from each subcommand in WST.
- **Used by:** WST
- **Details:** The input configuration files for WST are stored in the YAML file format. YAML is a human-readable file format that is well suited for storing hierarchical information. This information can be easily parsed and stored as common data types such as strings, scalars, lists and dictionaries by a range of programming languages. WST uses PyYAML to parse YAML files into Python data types. Basic YAML format specifications are listed below:
  - Each element of a YAML file is a key, value pair separated by a colon (key:value).
  - The key is the name given to the element, and the value is the data for that element.
  - The hierarchy of YAML files is maintained by outline indentation.
  - The number of spaces used to indent an element in the YAML file must be consistent across all elements at the same hierarchical level.
  - Nested data elements must be indented further than their preceding level.
  - Using tab for indentation is not recommended.
  - Optional blank lines can be added for readability.
  - Comments begin with the number sign (#) and must be separated from a key:value pair by space. Comments can start anywhere but are limited to a single line.
  - PyYAML automatically casts data types. For example, [123] is read as a list with a single integer value, ’123’ is read as a string, 123 is read as an integer, and 123.0 is read as a real number.
  - Strings do not require quotation (unless they could be cast as a number) and can contain spaces.
  - Lists are indicated with square brackets or hyphens. When using square brackets, the list is comma separated. When using hyphens, each entry of the list is on a new line.
  - Dictionaries are indicated with indentation or curly brackets and are used to define key: value pairs. Nested dictionaries define the hierarchical levels in the YAML file.

Additional information on YAML files can be found on the official YAML website `http://www.yaml.org`.

Select aspects of the `flushing` subcommand template configuration file are used as an example of the format of a YAML file. The full `flushing` subcommand template configuration file is shown in Figure 6.2. In the template, the top level key, denoted 'flushing', contains the following data:

```
# flushing configuration template
network:
  epanet file: Net3.inp                  # EPANET 2.00.12 network file name
scenario:
  location: [NZD]                        # Injection location: ALL, NZD or EPANET ID
  type: MASS                             # Injection type: MASS, CONCEN, FLOWPACED or SETPOINT
  strength: 100.0                        # Injection strength [mg/min or mg/L depending on
                                         #   type]
  species: null                          # Injection species, required for EPANET-MSX
  start time: 0                          # Injection start time [min]
  end time: 1440                         # Injection end time [min]
  tsg file: null                         # TSG file name, overrides injection parameters above
  tsi file: null                         # TSI file name, overrides TSG file
  signals: null                          # Signal files, overrides TSG or TSI files
  msx file: null                         # Multi-species extension file name
  msx species: null                      # MSX species to save
  merlion: false                         # Use Merlion as WQ simulator, true or false
impact:
  erd file: null                         # ERD database file name
  metric: [PE]                           # Impact metric
  tai file: Net3_bio.tai                 # Health impact file name, required for public health
                                         #   metrics
  response time: 0                       # Time [min] needed to respond
  detection limit: [0.0]                 # Thresholds needed to perform detection
  detection confidence: 1                # Number of sensors for detection
flushing:
  detection: [111, 127, 179]             # Sensor locations to detect contamination scenarios
  flush nodes:
    feasible nodes: NZD                  # Feasible flushing nodes
    infeasible nodes: NONE               # Infeasible flushing nodes
    max nodes: 2                         # Maximum number of nodes to flush
    rate: 800.0                          # Flushing rate [gallons/min]
    response time: 0.0                   # Time [min] between detection and flushing
    duration: 480.0                      # Flushing duration [min]
  close valves:
    feasible pipes: ALL                  # Feasible pipes to close
    infeasible pipes: NONE               # Infeasible pipes to close
    max pipes: 0                         # Maximum number of pipes to close
    response time: 0.0                   # Time [min] between detection and closing pipes
solver:
  type: StateMachineLS                   # Solver type
  options:                               # A dictionary of solver options
  threads: 1                             # Number of concurrent threads or function evaluations
  logfile: null                          # Redirect solver output to a logfile
  verbose: 0                             # Solver verbosity level
  initial points: []
configure:
  output prefix: Net3                    # Output file prefix
  output directory: SIGNALS_DATA_FOLDER  # Output directory
  debug: 0                               # Debugging level, default = 0
```

This subset of the the `flushing` subcommand template is refereed to as the flushing block. Instead of a single value assigned to 'flushing', the value is a dictionary containing a nested structure of additional key:value pairs.

The keys 'detection', 'flush nodes' and 'close valves' are all second level keys inside the flushing block. The location of these keys is often specified using the notation [flushing][detection], [flushing][flush nodes] and [flushing][close valves]. The second level keys must be indented using the same number of spaces and they must have unique names. The key [flushing][detection] is assigned to a list. Lists can be specified in one of two ways, using square brackets or using hyphen. The following two formats (separated by ---) are equivalent.

```
flushing:
  detection: [111, 127, 179] # square bracket notation, comma separated
---
flushing:
  detection: # hyphen notation, new line for each entry
  - 111
  - 127
  - 179
```

All template configuration files use square brackets to indicate where a list of input values can be used. If these input values include keywords, like NZD for non-zero demand nodes, this information is listed in the comment or in the user manual documentation for that specific YAML input parameter.

The options [flushing][flush nodes] and [flushing][close valves] are both assigned dictionaries. The data inside these second level keys contain additional key:value pairs. All keys within these dictionaries must be indented using the same number of spaces and have unique names. Two key:value pairs in the flushing flush nodes option are listed below:

```
flushing:
  flush nodes:
    feasible nodes: NZD
    max nodes: 2
```

Here, the [flushing][flush nodes][feasible nodes] option is set to the string NZD and the [flushing][flush nodes][max nodes] option is set to the integer 2. Note that there are two third-order keys named 'response time', however they do not share the same exact hierarchy, as shown below:

```
flushing:
  flush nodes:
    response time: 0.0
  close valves:
    response time: 0.0
```

YAML files can be written in a compact format that uses curly brackets to represent the hierarchical indentation of the nested dictonary. While this avoids issues with space indentation, this format is more difficult for the user to read. For example, the following two examples (separated by ---) are equivalent:

```
flushing:
  detection: [111, 127, 179]
---
{'flushing': {'detection': [111, 127, 179]}}
```

13.2 Cost File

- **Description:** Specifies the costs for installing sensors at different nodes throughout a network. This is the cost of installing one sensor at one particular node.
- **Format:** ASCII
- **Created by:** WST user
- **Used by:** `sp`
- **Details:** Each line of this file has the format:

```
   <EPANET node ID> <cost>
```

Nodes not explicitly enumerated in this file are assumed to have a cost of zero unless the ID __default is specified. For example to specify that all un-enumerated nodes have a cost of 1.0:

```
  __default 1.0
```

For example, the following cost file indicates that a sensor at node 1 has a cost of $100, a sensor at node 2 has a cost of $200 and sensors at all other nodes have a cost of $50.

```
    1       100
    2       200
  __default 50
```

13.3 ERD File

- **Description:** Provides a compact representation of all contamination scenario simulation results.
- **Format:** binary
- **Created by:** `tevasim`
- **Used by:** `sim2Impact`
- **Details:** The simulation data generator produces four output files containing the results of all contamination simulation scenarios. The database files include an index file (index.erd), a hydraulics file (hyd.erd) and a water quality file (qual.erd). The files are unformatted binary file in order to save disk space and computation time. They are not readable using an ordinary text editor.
- **Note:** The ERD file format replaced the TSO and SDX file formats, created by previous versions of `tevasim`, to extend the capability of `tevasim` for multi-species simulation using EPANET-MSX. While the `tevasim` subcommand produces only ERD files (even for single species simulation), the `sim2Impact` subcommand accepts both ERD and TSO file formats.

13.4 Impact File

- **Description:** For each contamination scenario, the impact file contains a list of all the locations (nodes) in the network where a sensor might detect contamination from a specific scenario.
- **Format:** ASCII
- **Created by:** `sim2Impact`
- **Used by:** `sp` and `evalsensor`
- **Details:** The first line of an impact file contains the number of incidents. The second line specifies the number of delays (always 1) and the delay time in minutes. Subsequent lines have the format

```
  <scenario-index> <node-index> <time-of-detection> <impact-value>
```

The scenario index is the index of contamination scenarios that were simulated. A scenario index maps to a line in the network scenariomap file, which is defined in Section 13.9. The node index is the index of a witness location for the incident. A node index maps to a line in the network nodemap file, which is defined in Section 13.8. The time of detection is in minutes. The value of the impacts are in the corresponding units for each impact metric. The different impact metrics in each line correspond to the different delays that have been computed.

13.5 Imperfect Junction Class File

- **Description:** Provides the mapping from EPANET node IDs to failure classes of different false-negative probabilities.
- **Format:** ASCII
- **Created by:** WST user
- **Used by:** `sp`
- **Details:** The imperfect junction class file provides the mapping from EPANET node IDs to failure classes of different false-negative probabilities as defined in a imperfect sensor class file (See Section 13.6 for information on imperfect sensor class files). The format of this file is:

  ```
  <node id> <failure class>
  <node id> <failure class>
  ....
  ```

  For example, node 1 is of class 2, node 2 is of class 1 and node 3 is of class 1:

  ```
  1 2
  2 1
  3 1
  ....
  ```

13.6 Imperfect Sensor Class File

- **Description:** Contains false-negative probabilities for different types of sensors. The false-negative probability defines the accuracy rate of the sensor (e.g., 50 percent of the time the sensor is providing a correct reading).
- **Format:** ASCII
- **Created by:** WST user
- **Used by:** `sp`
- **Details:** The file has format:

  ```
  <class id> <false-negative probability>
  <class id> <false-negative probability>
  ....
  ```

  For example, the following file defines a failure class 1 with a false-negative probability of 25 percent, and a failure class 2 with a false-negative probability of 50 percent:

  ```
  1 0.25
  2 0.5
  ....
  ```

13.7 Measurements File

- **Description:** Contains a list of measurements along with their corresponding time and EPANET node ID. This file can contain multiple node IDs and the measurement time is not required to be in order.
- **Format:** ASCII
- **Created by:** WST user or a water quality event detection system or a data acquisition system

- **Used by:** `inversion`
- **Details:** Each line of this file has the format:

```
   <EPANET node ID> <Time from beginning of simulation (sec) > <Binary yes/no measurement>
```

  An example measurements file is provided:

```
# Node_name      time    Cij


149 0 0
149 900 0
149 1800 0
149 2700 0
149 3600 0
149 4500 0
149 5400 0
```

13.8 Nodemap File

- **Description:** Provides a mapping from the indices used for sensor placement to the node IDs used within EPANET.
- **Format:** ASCII
- **Created by:** `sim2Impact`
- **Used by:** `evalsensor` and `sp`
- **Details:** Each line of this file has the format:

```
   <node-index> <EPANET node ID>
```

  This mapping is generated by the `sim2Impact` subcommand, and all sensor placement solvers subsequently use the node indices internally.

13.9 Scenariomap File

- **Description:** The scenariomap file provides auxiliary information about each contamination incident.
- **Format:** ASCII
- **Created by:** `sim2Impact`
- **Used by:** `evalsensor`
- **Details:** Each line of this file has the format:

```
   <node-index> <EPANET node ID> <source-type> <source-start-time> ...
                     <source-stop-time> <source-strength>
```

  The node index maps to the network nodemap file as described in Section 13.8, and the EPANET node ID provides this information. The source type is the injection mode for an scenario, e.g., flow-paced or fixed-concentration. The scenario source start and stop times are in minutes, and these values are relative to the start of the EPANET simulation. The source strength is the concentration of contaminant at the injection source.

13.10 Sensor Placement File

- **Description:** Describes one or more sensor placements.
- **Format:** ASCII
- **Created By:** `sp`
- **Used By:** `evalsensor`
- **Details:** Lines in a sensor placement file that begin with the # character are assumed to be comments. Otherwise, lines of this file have the format

```
  <sp-id> <number-of-sensors> <node-index-1> ...
```

  The sensor placement ID is used to identify sensor placements in the file. Sensor placements could have differing numbers of sensors, so each line contains this information. The node indices map to values in the nodemap file described in Section 13.8.

13.11 TAI File

- **Description:** Describes the information needed for assessing health impacts.
- **Format:** ASCII
- **Created by:** WST user
- **Used by:** `sim2Impact`
- **Details:** This file is required for health impact metrics, such as population exposed, population dosed and population killed. The following example can be copied directly into a text editor.

```
.; THREAT ASSESSMENT INPUT (TAI) FILE
; Data items explained below
; UPPERCASE items are non-modifiable keywords
; lowercase items are user-supplied values
; | indicates a selection
;----------------------------------------------------------------------
; INPUT-OUTPUT
; * TSONAME      - The location of the ERD or TSO file containing the results
;                  to analyze. This value is ignored when used in sim2Impact
;                  to specify parameters for the pe, pk or pd metrics.
; * TAONAME      - Name of threat assessment output (TAO) file. This value
;                  is ignored when used in sim2Impact to specify parameters
;                  for the pe, pk, or pd metrics.
; * SPECIES_NAME - The species name to analyze. This is optional - if it is not
;                  specified, the first species will be used. If the ERD database
;                  was generated by EPANET-MSX, the value MUST match one of
;                  the species specified in the MSX input file. If the ERD database
;                  was generated by EPANET 2.00.12, the value should be "species."
;                  If the TEVA-SPOT GUI generated the ERD database,
;                  the value MUST match the name specified in the GUI.
; * THRESHOLD    - The concentration threshold. All concentrations below
;                  value are set to 0. Only used in the the threatassess
;                  executable, not in sim2Impact
;----------------------------------------------------------------------
TSONAME       charstring
TAONAME       charstring
SPECIES_NAME  charstring
THRESHOLD     value
;----------------------------------------------------------------------
; DOSE-RESPONSE PARAMETERS
; * A - Function coefficient
; * M - Function coefficient
; * N - Function coefficient
; * TAU - Function coefficient
; * BODYMASS - Exposed individual body mass (kg)
; * NORMALIZE - Dose in mg/kg (YES) or mg (NO)
; * BETA - Beta value for probit dose response model
; * LD50 - LD50 or ID50 value for the agent being studied
; * TYPE - Either PROBIT or OLD depending on the dose response equation
;          to be used. If it is PROBIT, only the LD50 and BETA values
;          need to be specified, and if it is OLD, the A, M, N and TAU
;          values need to be specified. The BODYMASS and NORMALIZE
;          apply to both equations.
;----------------------------------------------------------------------
DR:A          value
DR:M          value
DR:N          value
DR:TAU        value
BODYMASS      value
NORMALIZE     YES|NO
DR:BETA       value
DR:LD50       value
DR:TYPE       probit | old
;----------------------------------------------------------------------
; DISEASE MODEL
; * LATENCYTIME - Time from exposed to symptoms (hours)
; * FATALITYTIME - Time from symptoms till death (hours)
; * FATALITYRATE - Fraction of exposed population that die
;----------------------------------------------------------------------
LATENCYTIME   value
FATALITYTIME  value
FATALITYRATE  value
;----------------------------------------------------------------------
; EXPOSURE MODEL
; * DOSETYPE      - TOTAL = Total ingested mass
; * INGESTIONTYPE - DEMAND = Ingestion probability proportional to demand
;                   ATUS RANDOM = ATUS ingestion model, random volume
;                                 selection from volume curve
;                   ATUS MEAN   = ATUS ingestion model, mean volume of value
;                   FIXED5 RANDOM = 5 fixed ingestion times (7AM, 9:30AM, Noon, 3PM, 6PM),
;                                 random volume selection from volume curve
;                   FIXED5 MEAN = 5 fixed ingestion times (7AM, 9:30AM, Noon, 3PM, 6PM),
;                                 mean volume of value
; * INGESTIONRATE - Volumetric ingestion rate (liters/day) - used for DEMAND,
;                   ATUS MEAN and FIXED5 MEAN
;----------------------------------------------------------------------
DOSETYPE      TOTAL
INGESTIONTYPE DEMAND | ATUS RANDOM | ATUS MEAN | FIXED5 RANDOM | FIXED5 MEAN
INGESTIONRATE value
;----------------------------------------------------------------------
; POPULATION MODEL
; * POPULATION FILE  - File name that contains the node-based
;                      population. The format of the file is simply
;                      one line per node with the node ID and the
;                      population value for that node.
; * POPULATION DEMAND  - Per capita usage rate (flow units/person).
;                        The population will be estimated by demand.
;----------------------------------------------------------------------
POPULATION    FILE | DEMAND  value
;----------------------------------------------------------------------
; DOSE OVER THRESHOLD MODEL
; * DOSE_THRESHOLDS - The dose over each threshold specified will be
;                     computed and output to the TAO file.
; * DOSE_BINDATA - Specifies the endpoints of bins to tally the number
;                  of people with a dose between the two endpoints.
;                  Values can be either dose values or response values -
;                response values are converted to dose values using the
;                dose-response data specified in this file and are indicated
;                on this line by the keyword RESPONSE. Dose values are
;                IDENTIFIED by the keyword DOSE.
;----------------------------------------------------------------------
DOSE_THRESHOLDS value1 ... value_n
```

13.12 TSG File

- **Description:** Specifies how an ensemble of EPANET 2.00.12 contamination scenario simulations will be performed.
- **Format:** ASCII
- **Created by:** WST user
- **Used by:** `tevasim`
- **Details:** Each line of a TSG file specifies injection location(s), species (optional), injection mass and the injection time-frame:

```
   <injection-location> <injection-type> <specie> <injection-mass> <start-time> <end-time>
```

If <specie> is included, the `tevasim` subcommand uses EPANET-MSX. The simulation data generator uses the specifications in the TSG file to construct a separate threat simulation input (TSI) file that describes each individual contamination scenario in the ensemble. Each line in the TSG file uses a simple language that is expanded to define the ensemble. The entire ensemble is comprised of the cumulative effect of all lines in the TSG file. The TSG file is an optional file, since the ensemble of contamination scenarios can be specified in the configuration file.

```
<Src1><SrcN> <SrcType> <SrcSpecie> <SrcStrngth> <Start> <Stop>

<Srci>:       A label that describes the ith source location of an N-source ensemble.
              This can be either: 1) An EPANET node ID identifying one node
              where the contaminant is introduced, 2) ALL, denoting all nodes
              (excluding tanks and reservoirs), 3) NZD, denoting all nodes with
              non-zero demands. This simple language allows easy specification of
              single- or multi-source ensembles. [Character strings]
<SrcType>:    The source type, one of: MASS, CONCEN, FLOWPACED, SETPOINT (see EPANET 2.00.12
              user manual for information about these types of water quality sources).
              [Character string]
<SrcSpecie>:  The character ID of the water quality species added by the source. This
              parameter must be omitted when using executables built from the standard
              EPANET 2.00.12 distribution. [Character string]
<SrcStrngth>: The strength of the contaminant source (see EPANET 2.00.12 documentation for
              the various source types).
<Start>:      The time, in seconds, measured from the start of simulation, when the
              contaminant injection is started. [Integer]
<Stop>:       The time, in seconds, measured from the start of simulation, when the
              contaminant injection is stopped. [Integer]

Examples:

One scenario with a single injection at node ID 10, mass rate of 5 mg/min of species
SPECIE1, start time of 0, and stop time of 1000:
10     MASS    SPECIE1    5     0     1000

Multiple scenarios with single injections at all non-zero demand nodes:
NZD   MASS    SPECIE1   5     0     1000

Multiple scenarios with two injections, one at node ID 10, and the other at all
non-zero demand nodes (NZD):
```

```
10     NZD     MASS     SPECIE1   5     0     1000

Multiple scenarios with three injections, at all combinations of all nodes
(if there are N nodes, this would generate N^3 scenarios for the ensemble):
ALL     ALL     ALL   MASS   SPECIE1   5     0     1000

Note: this language will generate scenarios with repeat instances of injection node
locations (e.g., ALL   ALL would generate one scenario for node i and j, and another
identical one for node j and i). Also, it will generate multi-source scenarios with
the same node repeated. In this latter case, the source mass rate at the repeated
node is the mass rate specified in <SrcStrngth>.
```

13.13 TSI File

- **Description:** Specifies how an ensemble of EPANET 2.00.12 contamination scenario simulations will be performed.
- **Format:** ASCII
- **Created by:** `tevasim` or WST user
- **Used by:** `tevasim`
- **Details:** The TSI file is generated as output from the `tevasim` subcommand and would not normally be used, but it is available after the run for reviewing each scenario that was generated for the ensemble. The TSG file is essentially a short hand for generation of the more cumbersome TSI file. Each record in the TSI file specifies the unique attributes of one contamination scenario. The number of scenarios does not have a restriction.

```
<NodeID1> <SrcTypeIDX1> <SrcSpecieIDX1> <SrcStrngth1> <Start1> <Stop1>  <NodeIDN>...
 <SrcTypeIDXN> <SrcSpecieIDXN> <SrcStrngthN> <StartN> <StopN>

<NodeIDi>:              EPANET ID identifying the ith node where the contaminant is
                        introduced.  [Character string]
<SrcTypeIDXi>:          The EPANET source type index of the ith contaminant source.
                        Each EPANET source type is associated with an integer index
                        (see EPANET 2.00.12 toolkit documentation for reference). [Integer]
<SrcSpecieIDXi>:        The EPANET species index of the ith contaminant source. [Integer]
<SrcStrngthi>:          The strength of the ith contaminant source (see EPANET 2.00.12
                        documentation for description of sources). This value
                        represents the product of contaminant flow rate and
                        concentration. [Real number]
<Starti>:               The time, in seconds, measured from the start of simulation,
                        when the ith contaminant injection is started. [Integer]
<Stopi>:                The time, in seconds, measured from the start of simulation,
                        when the ith contaminant injection is stopped. [Integer]

One water quality simulation will be run for each scenario specified in the threat
simulation input (TSI) file. For each such simulation, the source associated with each
contaminant location <NodeIDi>, i=1,,N will be activated as the specified type source,
and all other water quality sources disabled. If a source node is specified in the
EPANET 2.00.12 input file, the baseline source strength and source type options will be ignored,
```

13.14 Weight File

- **Description:** Specifies the weights for contamination incidents.
- **Format:** ASCII
- **Created by:** WST user

- **Used by:** `sp`
- **Details:** Each line of this file has the format:

  ```
  <scenario-ID> <weight>
  ```

  Scenarios not explicitly enumerated in this file are assumed to have a weight of zero unless the ID __default is specified. For example, to specify that all un-enumerated scenarios have a weight of 1.0:

  ```
  __default 1.0
  ```

Chapter 14

Executable Files

This chapter describes the different executable files that can be used outside of WST to evaluate different sensor network designs, and to reduce the size of the sensor placement problem by filtering impacts, aggregating impacts or skeletonizing the water distribution network model. In addition, an executable file to create a simulated measurements file for sensor locations in a water distribution network model is also described.

14.1 evalsensor

The `evalsensor` executable is used to compute information about the impact of contamination incidents for one or more sensor network designs. The `evalsensor` executable takes a sensor network design in a sensor placement file (see File Formats Section 13.10 for more detail) and evaluates them using data from an impact file or a list of impact files (see File Formats Section 13.4). This executable measures the performance of each sensor network designs with respect to the set of possible contamination scenarios.

Section 12.2.4 provides more information and an example application of this executable.

14.1.1 Usage

Usage with a specific sensor network design:

```
   evalsensor [options...] <sensor-file> <impact-file1> [<impact-file2>...]
```

Usage without a sensor network design:

```
   evalsensor [options...] none <impact-file1> [<impact-file2>...]
If none, is specified, then evalsensor will evaluate impacts without any sensors.
```

14.1.2 Options

```
     --all-locs-feasible
     A boolean flag to indicate that all locations are treated as feasible.

     --costs=<filename>
     The name of a file that contains the cost information for each node in the network.
         For more details about the cost file, see File Formats Section 13.2.

     --debug
     A boolean flag to add output information about each incident.

     --format=<type>
     The type of output that the evaluation will generate:
                cout -  Generates output that is easily read. (default)
                xls -   Generates output that is easily imported into a MS Excel spreadsheet.
                xml -   Generates an XML-formatted output to communicate
```

```
                           with the TEVA-SPOT GUI. (not currently supported)

--gamma=<num>
The fraction of the tail distribution used to compute the VaR and TCE
    performance measures. (default is 0.05)

    -h, --help
    A boolean flag to display usage information.

    --incident-weights=<filename>
    The name of a file that contains the weights of the different contamination incidents.
    For more details about the weights file, see File Formats Section 13.14.

    --nodemap=<filename>
    The name of a file that contains the node map information for translating sensor placement
    indices to EPANET node IDs. For more details about the nodemap file, see File Formats
    Section 13.8.

    -r, --responseTime=<num>
    The number of minutes that are needed to respond to the detection of
    contamination. As the response time increases, the impact increases
    because the contaminant affects the network for a greater length of
    time.

    --sc-probabilities=<filename>
    The name of a file that contains the probability of detection for each sensor category.
    For more details about the imperfect sensor class file, see File Formats Section
    13.6.

    --scs=<filename>
    The name of a file that contains the sensor category information for each possible
    sensor location in the network. For more details about the imperfect junction class file,
    see File Formats Section 13.5.

    --version
    A boolean flag to display version information.

    Note: Options like reponseTime can be specified with the syntax
    --responseTime 10.0 or --responseTime=10.0.
```

14.1.3 Arguments

```
<sensor-file>
A sensor placement file that contains one or more sensor network designs
    that will be evaluated. If none, is specified, then evalsensor will evaluate
    impacts without any sensors.

<impact-file>
A impact file that contains the impact data concerning the simulated contamination
    incidents. If one or more impact files are specified, then evaluations are
    performed for each impact separately.
```

14.2 filter_impacts

The `filter_impacts` script filters out the low-impact incidents from an impact file. The `filter_impacts` command reads an impact file, filters out the low-impact incidents, rescales the impact values and outputs another impact file.

14.2.1 Usage

```
filter_impacts [options...] <impact-file> <out-file>
```

14.2.2 Options

```
--threshold=<val>
The contamination incidents with undetected impacts above a specified threshold should be kept.

--percent=<num>
The percentage of contamination incidents with the worst undetected impact that should be kept.

--num=<num>
The number of contamination incidents with the worst undetected impact that should be kept.

--rescale
Rescale the impacts using a log10 scale.

--round
Round input values to the nearest integer.
```

14.2.3 Arguments

```
<impact-file>
The input impact file.

<out-file>
The output impact file.
```

14.3 measuregen

The `measuregen` executable is used to create a simulated measurements file for sensor locations in a water distribution network model. A node-time-concentration list is obtained by water quality simulations performed in Merlion. The MEASURE.dat file contains the node-time-concentration list and can be used to perform source inversion.

14.3.1 Usage

```
   measuregen [options...] <required network option> <required scenario option> <sensor-file>
```

14.3.2 Options

```
Data format option:
     --output-prefix=<filename>
     The name to add to all output files.

EPANET input file options:
     --quality-timestep-minutes=<num>
     The size of the water quality time step used by Merlion to perform the water quality simulations.
     When this value is specified, it overrides the value in the EPANET 2.00.12 input file.

     --simulation-duration-minutes=<num>
     The length of water quality simulation used to build the Merlion water quality model.
     When this value is specified, it overrides the value in the EPANET 2.00.12 input file. This is useful
     when the length of the simulation specified in the EPANET 2.00.12 input file is longer than the time
     horizon in which the sensor measurements are needed. For instance, if the simulation duration
     in the EPANET 2.00.12 input is seven days, but sensor measurements are only needed for the first three
     days of the simulation. This option reduces the memory required to build the Merlion water
     quality model.

Label options:
     --custom-label-map=<filename>
     The name of a file which maps EPANET node names to custom labels. All data files will be written
     using these custom labels.

     --output-merlion-labels
     Node names will be translated into integer node IDs to reduce file sizes for large networks.
     A node map is provided to map node IDs back to node names (MERLION_LABEL_MAP.txt). This option
     is overridden by the --custom-label-map flag.

Noise options:
     --FNR=<num>
     The false negative rate to apply to all sensors.

     --FPR=<num>
     The false positive rate to apply to all sensors.

     --scale=<num>
     The scaling value to add noise to the base demand.

     --seed=<num>
     The seed to generate the random number used at the moment of adding noise.

Other options:
     --disable-warnings
     A boolean flag to disables printing of warning statements to stderr.

     --enable-logging
     A boolean flag to generate a log file with verbose runtime information.

     -h, --help
     A boolean flag to display usage information.
     --ignore-merlion-warnings
     A boolean flag to ignore warnings about unsupported features of Merlion.

     -v, --version
     A boolean flag to display version information.

Save options:
     --epanet-rpt-file
     A boolean flag to save output file generated by EPANET 2.00.12 during hydraulic simulations.

     --merlion-save-file
     A boolean flag to save the text file defining the Merlion water quality model.

Time options:
     --concentrations
     The concentration values will be printed in the measurement file.

     --decay-const=<num>
     The value for the first-order decay coefficient(1/min). The default value is taken from EPANET 2.00.12
     input file.

     --measures-per-hour=<num>
     The number of measurements to take per hour. The default value is 60.

     --start-sensing-time=<num>
     The time to start taking measurements. This value is in minutes.

     --stop-sensing-time=<num>
     The time to stop taking measurements. This value is in minutes.

     --threshold=<num>
     The value of concentrations above which the measurements are positive. The default value is 0.0.
```

14.3.3 Arguments

```
     <required network option>
     --inp=<filename>
     The name of the EPANET 2.00.12 network file.

     --wqm=<filename>
     The name of the Merlion water quality model (wqm) file.

     <required scenario option>
     This argument defines the injection incidents to simulate in order
     to obtain measurements at the sensor locations. Three options are
     available to define the injection incidents.

     --scn=<filename>
     The name of the SCN file for specifying the injection incidents.

     --tsg=<filename>
     The name of the TSG file for specifying the injection incidents.

     --tsi=<filename>
     The name of the TSI file for specifying the injection incidents.

     --tsi-species-id=<name>
     (*optional) The single TSI species id to use in each scenario by Merlion. All other species
     will be ignored. If this option is not used and multiple species ids are in the TSI
     file, an error will occur.

     <sensor node file>
     A file with a list of sensor node names.
```

14.4 scenarioAggr

The `scenarioAggr` executable takes an impact file and produces an aggregated impact file. The `scenarioAggr` executable reads an impact file, finds similar incidents, combines them and writes out another impact file. The convention is to append the string aggr to the output.

The following files are generated during the execution of `scenarioAggr`, assuming that the input was named network.impact:

- aggrnetwork.impact - The new impact file.
- aggrnetwork.impact.prob - The probabilities of the aggregated incidents. These are non-uniform, so any solver must recognize incident probabilities.

Not all of the solvers available in the `sp` command can perform optimization with aggregated impact files. In particular, the heuristic GRASP solver does not currently support aggregation because it does not use contamination incident probabilities. The Lagrangian and PICO solvers support contamination incident aggregation. However, initial results suggest that although the number of contamination incidents is reduced significantly, the number of impacts might not be, and solvers might not run much faster.

14.4.1 Usage

```
scenarioAggr --numEvents=<num_incidents> <impact file>
```

14.4.2 Options

```
--numEvents=<number>
The number of contamination incidents that should be aggregated.
```

14.4.3 Arguments

```
<impact-file>
The input impact file.
```

14.5 spotSkeleton

The skeletonizer, spotSkeleton, reduces the size of a network model by grouping neighboring nodes based on the topology of the network and pipe diameter threshold. Nodes that are grouped together form a new node, often referred to as a supernode. The spotSkeleton executable requires an EPANET 2.00.12 INP network file and a pipe diameter threshold. The executable creates a skeletonized EPANET 2.00.12 INP network file and map file. The map file defines the nodes that belong to each supernode.

The spotSkeleton executable includes branch trimming, series pipe merging and parallel pipe merging. A pipe diameter threshold determines candidate pipes for skeleton steps. The spotSkeleton executable maintains pipes and nodes with hydraulic controls as it creates the skeletonized network. It performs series and parallel pipe merges if both pipes are below the pipe diameter threshold, calculating hydraulic equivalency for each merge based on the average pipe roughness of the joining pipes. For all merge steps, the larger diameter pipe is retained. For a series pipe merge, demands (and associated demand patterns) are redistributed to the nearest node. Branch trimming removes deadend pipes smaller than the pipe diameter threshold and redistributes demands (and associated demand patterns) to the remaining junction. The spotSkeleton executable repeats these steps until no further pipes can be removed from the network. The spotSkeleton executable creates an EPANET-compatible skeletonized 2.00.12 network INP file and a map file that contains the mapping of original network model nodes into skeletonized supernodes.

Under these skeletonization steps, there is a limit to how much a network can be reduced based on its topology, e.g., number of deadend pipes, or pipes in series and parallel. For example, sections of the network with a loop, or grid, structure will not reduce under these skeleton steps. Additionally, the number of hydraulic controls influences skeletonization, as all pipes and nodes associated with these features are preserved.

Commercial skeletonization codes include Haestad Skelebrator, MWHSoft H2OMAP and MWHSoft InfoWater. To validate the spotSkeleton executable, its output was compared to the output of MWHSoft H2OMAP and Infowater. Input parameters were chosen to match spotSkeleton options. Pipe diameter thresholds of 8 inches, 12 inches and 16 inches were tested using two large networks. MWHSoft and WST skeletonizers were compared using the Jaccard index. The Jaccard index measures similarity between two sets by dividing the intersection of the two sets by the union of the two sets. In this case, the intersection is the number of pipes that are either both removed or not removed by the two skeletonizers, and the union is the number of all pipes in the original network. If the two skeletonizers define the same supernodes, the Jaccard index equals 1. Skeletonized networks from the MWHSoft and WST skeletonizers resulted in a Jaccard index between 0.93 and 0.95. Thus, the spotSkeleton executable is believed to be a good substitute for commercial skeletonizers.

14.5.1 Usage

```
spotSkeleton <input inp file> <pipe diameter threshold> <output inp file> <output map file>
```

14.5.2 Arguments

```
<input inp file>
The input EPANET 2.00.12 INP file to be skeletonized.

<pipe diameter threshold>
The pipe diameter threshold that determines which pipes might be skeletonized.

<output inp file>
The output EPANET 2.00.12 INP file created after skeletonization.

<output map file>
The output map file that contains the mapping of original network nodes to
skeletonized supernodes.
```

References

Adams, B. M., Ebeida, M. S., Eldred, M. S., Jakeman, J. D., Swiler, L. P., Bohnhoff, W. J., Dalbey, K. R., Eddy, J. P., Hu, K. T., and Vigil, D. M. (2013). Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantifcation, and sensitivity analysis. Technical Report SAND2010-2183, Sandia National Laboratories. Available at http://dakota.sandia.gov/documentation.html.

Berry, J., Hart, W. E., Phillips, C. E., Uber, J. G., and Watson, J.-P. (2006). Sensor placement in municipal water networks with temporal integer programming models. *J. Water Resources Planning and Management*, 132(4):218–224.

Bureau of Labor Statistics and U.S. Census Bureau (2005). American time use survey user's guide 2003 to 2004. Technical report, Washington DC. Available at http://www.bls.gov/news.release/archives/atus_09142004.pdf.

Daskin, M. (1995). Wiley, New York.

Davis, M. and Janke, R. (2008). Importance of exposure model in estimating impacts when a water distribution system is contaminated. *J. Water Resources Planning and Management*, 134(5):449–456.

De Sanctis, A., Shang, F., and Uber, J. (2009). Real-time identification of possible contamination sources using network backtracking methods. *Journal of Water Resources Planning and Management*, 136:444–453.

Elloumi, S., Labbé, M., and Pochet, Y. (2004). *INFORMS Journal on Computing*, 16:83–94.

EPA, U. S. (2004). Response protocol toolbox: planning for and responding to drinking water contamination threats and incidents. Technical report, U.S. Environmental Protection Agency, Office of Water, Office of Ground Water and Drinking Water, Washington, D.C. Available at http://water.epa.gov/infrastructure/watersecurity/upload/2004_11_24_rptb_response_guidelines.pdf.

EPA, U. S. (2011). Teva-spot toolkit and user's manual. Technical report, U.S. Environmental Protection Agency. Available at http://cfpub.epa.gov/si/si_public_file_download.cfm?p_download_id=514412.

EPA, U. S. (2013a). Epanet-rtx, real-time extension for the epanet toolkit, at open water analytics. Technical report, U.S. Environmental Protection Agency. Available at http://openwateranalytics.github.io/epanet-rtx/index.html.

EPA, U. S. (2013b). Water security initiative. Technical report, U.S. Environmental Protection Agency. Available at http://water.epa.gov/infrastructure/watersecurity/lawregs/initiative.cfm.

Fourer, R., Gay, D. M., and Kernighan, B. W. (2002). *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole, Pacific Grove, CA, 2nd edition.

Geib, C., Taxon, T., and Hatchett, S. (2011). Epanet results database (erd), user's guide, version 1.00.00. Technical report.

Hart, D. B. and McKenna, S. A. (2012). Canary user's manual, version 4.3.2. Technical Report EPA/600/R/08/040B, Washington, D,C.: U.S. Environmental Protection Agency. Available at http://cfpub.epa.gov/si/si_public_file_download.cfm?p_download_id=513254.

Hart, W. E., Laird, C., Watson, J., and Woodruff, D. (2012). *Pyomo - Optimization Modeling in Python*. Springer, 1st edition.

Hatchett, S., Uber, J., Boccelli, D., Haxton, T., Janke, R., Kramer, A., Matracia, A., and Panguluri, S. (2011). Real-time distribution system modeling: development, application, and insights. In *In Proc. Eleventh International Conference on Computing and Control for the Water Industry, Sept. 2011, Exeter, UK*.

Janke, R., Morley, K., Uber, J., and Haxton, T. (2011). Real-time modeling for water distribution system operation: Integrating security developed technologies with normal operations. In *In Proc. AWWA Distribution Systems Symposium and Water Security Conference, Sept. 2011, Nashville, TN*.

Klise, K., Phillips, C., and Janke, R. (2013). Two-tiered sensor placement using skeletonized water distribution network models. *Journal of Infrastructure Systems*, (10.1061/(ASCE)IS.1943-555X.0000156).

Klise, K., Siirola, J., Hart, D., Hart, W., Phillips, C., Haxton, T., Murray, R., Janke, R., Taxon, T., Laird, C., Seth, A., Hackebeil, G., McGee, S., and Mann, A. (2015). Water security toolkit user manual version 1.3. Technical Report SAND2015-9735, Sandia National Laboratories.

Mann, A., Hackebeil, G., and Laird, C. (2012a). Explicit water quality model generation and rapid multi-scenario simulation. *J. Water Resources Planning and Management*, (10.1061/(ASCE)WR.1943-5452.0000278).

Mann, A., McKenna, S., Hart, W., and Laird, C. (2012b). Real-time inversion in large-scale water networks using discrete measurements. *Computers & Chemical Engineering*, 37:143–151.

Mirchandani, P. and Francis, R., editors (1990). *Discrete Location Theory*. John Wiley and Sons, Toronto, Canada.

Murray, R., Adcock, N., Rice, G., Uber, J., and Hatchett, S. (2011). Predicting pathogen survival when introduced into a water distribution system with growth medium. *In Proc. Water Quality Technology Conference, Nov. 2011, Phoenix, AZ*.

Murray, R., Haxton, T., Janke, R., Hart, W. E., Berry, J., and Phillips, C. (2010). Sensor network design for drinking water contamination warning systems: A compendium of research results and case studies using the teva-spot software. Technical Report EPA/600/R-09/141, Cincinnati, OH Office of Research and Development, National Homeland Security Research Center, Water Infrastructure Protection. Available at http://cfpub.epa.gov/si/si_public_file_download.cfm?p_download_id=498251.

Murray, R., Uber, J., and Janke, R. (2006). Model for estimating acute health impacts from consumption of contaminated drinking water. *J. Water Resources Planning and Management: Special Issue on Drinking Water Distribution Systems Security*, 132(4):293–299.

Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *J. of Banking and Finance*, 26(7):1443–1471.

Rossman, L. A. (2000). Epanet 2 users manual. Technical report, Environmental Protection Agency. Available at http://nepis.epa.gov/Adobe/PDF/P1007WWU.pdf.

Shang, F., Uber, J., and Polycarpou, M. (2002). Particle backtracking algorithm for water distribution system analysis. *Journal of Environmental Engineering*, 128:441–450.

Shang, F., Uber, J., and Rossman, L. (2011). Epanet mutli-species extension user's manual. Technical Report EPA/600/S-07/021, USEPA. Available at http://cfpub.epa.gov/si/si_public_file_download.cfm?p_download_id=500759.

Topaloglou, N., Vladimirou, H., and Zenios, S. (2002). CVaR models with selective hedging for international asset allocation. *J. of Banking and Finance*, (26):1535–1561.

U.S. Geological Survey (2004). Estimated use of water in the united states in 2000. Technical report, U.S. Geological Survey. Available at `http://pubs.usgs.gov/circ/2004/circ1268/pdf/circular1268.pdf`.

Watson, J.-P., Hart, W. E., and Murray, R. (2006). Formulation and optimization of robust sensor placement problems for contaminant warning systems. In *In Proc. Water Distribution System Symposium*.

Watson, J.-P., Murray, R., and Hart, W. E. (2009). Formulation and optimization of robust sensor placement problems for drinking water contamination warning systems. *Jour. Infrastructure Systems*, 15(4):330–339.

EPA
United States
Environmental Protection
Agency

Office of Research and Development (8101R)
Washington, DC 20460