CIVL 555 - Final Project Discussion ¶

Name	Student Number	Date
Dan Kovacek	35402767	17 April 2020

Overview

The focus of this presentation is the study:

> *Sensor Placement in Water Networks Using a Population-Based Ant Colony Optimization Algorithm* (Diwold, Ruhnke, and Middendorf 2010).

The basic underlying question addressed in the paper asks:

> How do we go about placing sensors to best protect a municipal water distribution system from (un)intentional contamination?

The format of this presentation is an interactive Python notebook (using Jupyter (https://chama.readthedocs.io/en/latest/overview.html)), and the aim is to develop an intuition of the general approach to the optimal sensor placement problem as it is applied to the contamination detection problem in water distribution networks. The paper I chose to focus on for this discussion applies a genetic algorithm (GA) based on the Ant Colony Optimization (ACO) algorithm to the problem of optimal sensor placement for the contaminant detection problem.

Case Study

Like several other studies, (Diwold, Ruhnke, and Middendorf 2010) uses two municipal water networks (MWN) provided by the United States Envirionmental Protection Agency (USEPA) as benchmark examples to demonstrate the performance of the ACO algorithm developed in their study. As noted in the paper, both MWNs evaluated in the study are accessible online (http://www.projects.ex.ac.uk/cws/) (with a little digging), and the network selected for demonstration herein represents the water distribution network of Richmond, Virginia, USA. The Richmond MWN features 872 nodes and 957 connecting pipes.

Questions to Consider

1. What are different ways in which the objective(s) could be stated?
   - What are we trying to accomplish?

1. How would you evaluate the effectiveness of a solution in order to compare it with others?
   - What makes a good solution?

1. Where might we look to determine the largest sources of uncertainty that limit the power of the approach?
   - What aspect of the approach is furthest from reality?

In [104]:
```
# python package imports
import wntr
import chama
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
```

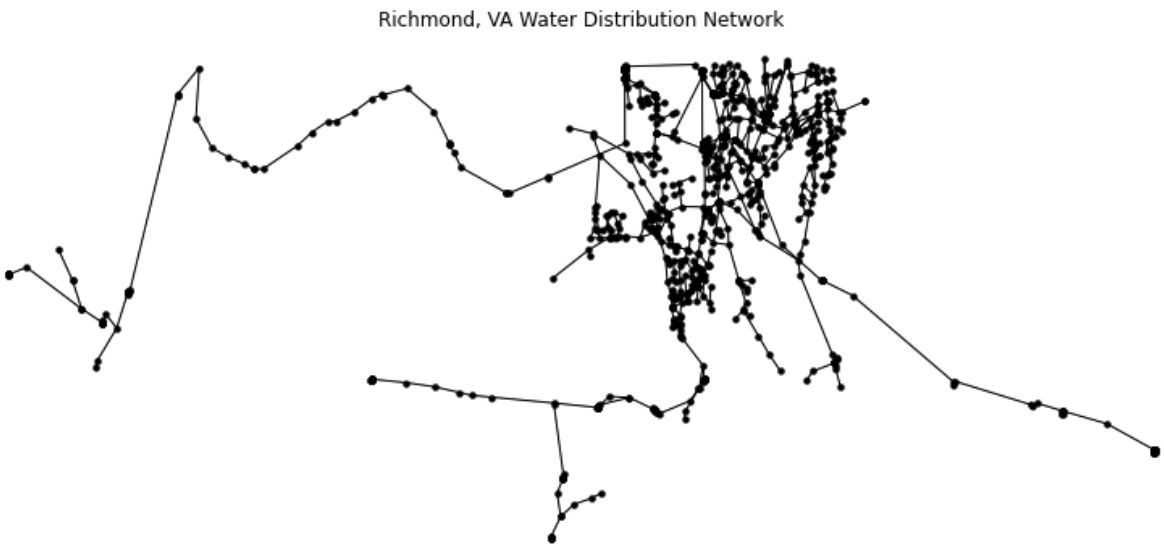
Import and Visualize the Water Distribution Network

In [8]:
```
# import the network graph representation of the MWN
# (the 'skeletion' is a smaller network for code development)
# filename = 'Richmond_skeleton.txt'
filename = 'networks/Richmond.txt'
wn = wntr.network.WaterNetworkModel(filename)
```

In [9]:
```
wn_description = wn.describe(level=2)
for p in wn_description:
    print('{}, '.format(p), wn_description[p])
```

```
Nodes,  {'Junctions': 865, 'Tanks': 6, 'Reservoirs': 1}
Links,  {'Pipes': 949, 'Pumps': {'Head': 7, 'Power': 0}, 'Valves': {'PRV':
1, 'PSV': 0, 'PBV': 0, 'TCV': 0, 'FCV': 0, 'GPV': 0}}
Patterns,  21
Curves,  {'Pump': 7, 'Efficiency': 7, 'Headloss': 0, 'Volume': 0}
Sources,  0
Controls,  16
```

In [69]:
```
# visualize the network
net_plot = wntr.graphics.plot_network(wn, #node_attribute='elevation',
                                      title="Richmond, VA Water Distributio
n Network")
# plt.rcParams["figure.figsize"] = [14,6]
plt.show()
```



Define the Objective Functions

In the case of (Diwold, Ruhnke, and Middendorf 2010), the decision space corresponds to placement of a set number of sensors in the network to optimize the following objectives:

1. **Minimize the detection time of contaminations (Z_1)**, defined as the elapsed time between the start of the contamination event and the detection by a sensor, assuming ideal sensor operation.
$$Z_1(S, C) = \frac{1}{D_S^C} \sum_{C \in D_S^C} d(C, S)$$
2. **Minimize the rate of non-detection (Z_2)**, defined as the number of contamination scenarios for which the given network cannot detect the contamination event.
$$Z_2(S, C) = 1 - \frac{|D_S^C|}{|C|}$$

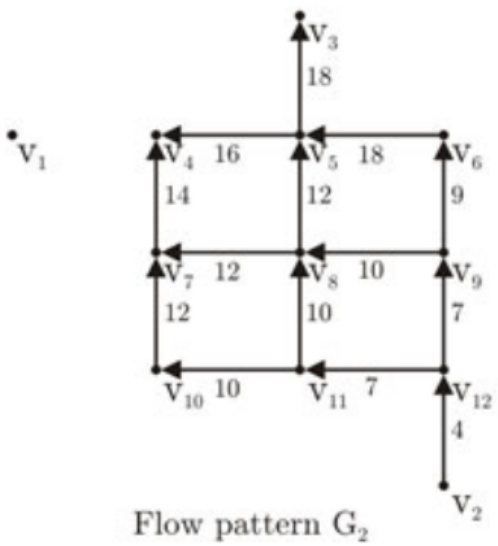
It is not initially apparent, but the two goals are in opposition, so movement in the decision space is necessarily a tradeoff of better performance of one criterion for worse performance of another criterion.

Novel Approach

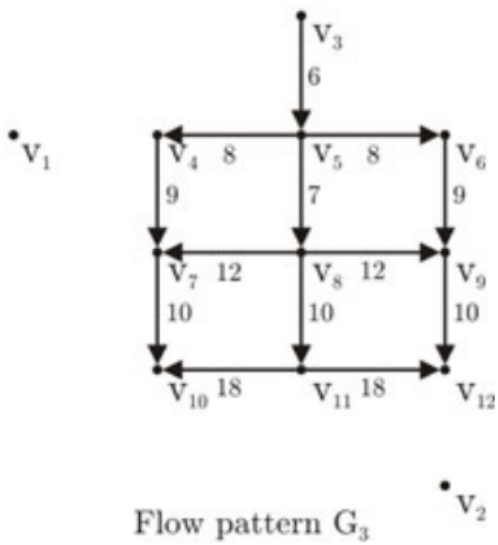
Sensor placement for contamination detection in a MWN is well studied, so what is novel about the approach used in (Diwold, Ruhnke, and Middendorf 2010)?

- **Flow patterns**: the research recognizes that changing demand patterns have an effect on the transport of contaminants. Demand patterns change on different timescales. Think of a school's water use throughought a single day, and throughout a year. Incorporating changing flow patterns adds to the computational complexity considerably, but captures a much more realistic picture of the hydraulics of the system compared to assuming a single quasi-steady-state flow pattern.
- **WSP-PACO**: a unique variation of the ACO is developed in the paper, which is itself a variation on a "Population based ACO", or PACO, developed in previous work to specifically address the ACO for multiobjective optimization.

Below is a diagram showing how different flow patterns are represented in the network graph formulation. Source: (Diwold, Ruhnke, and Middendorf 2010).



Flow pattern G_2



Flow pattern G_3

Ant Colony Optimiztion (ACO)

The ACO is an algorithm that mimics the foraging behaviour of ant colonies in nature. Individual ants can search the decision space near the current set of optimal solutions to make refinements on the existing solution (referred to as local heuristics), or they can search farther away to escape the local optima and find a better solution (pheromone information).

(Population Based) Ant Colony Optimization (PACO)

The ACO is limited to single-objective optimization problems. An approach to address multi-objective optimization problems was proposed by (Guntsch et al. 2003) wherein instead of the set of ants only remembering one best solution each generation, a 'population' of best solutions describing the non-dominated Pareto set is constructed. The pheromone information gets distributed to this population (Pareto set), and the population is updated when a newly encountered solution is found to dominate solutions in the population. The newly dominated solution is then removed altogether from having influence on the search in future generations.

Let's Try It!

How large is this decision space?

As shown below, there are 400 billion combinations of 5 sensors possible in the Richmond, VA network.

In (Diwold, Ruhnke, and Middendorf 2010), the entire decision space is not solved. Rather, two sets of 2000 randomly selected scenarios are used

In [108]:
```
def binom(n, k):
    return math.factorial(n) // math.factorial(k) // math.factorial(n - k)

number_of_nodes = len(wn.junction_name_list)
number_of_sensors = 5
decision_space_size = binom(number_of_nodes, number_of_sensors)
print('There are {:.1e} possible decisions in the decision space.'.format(d
ecision_space_size))
```

```
There are 4.0e+12 possible decisions in the decision space.
```

Set up the parameters for modelling the contamination events

Contamination events in (Diwold, Ruhnke, and Middendorf 2010) were modelled after those developed in a previous study, which are described as follows (Ostfeld et al. 2008):

> ...an ensemble of contamination scenarios sampled from a statistical distribution; the probability distribution of contamination events is described herein. Contaminant intrusions occurred at network nodes, with an injection flow rate of 125 L/h, contaminant concentration of 230,000 mg/L, and an injection duration of 2h.

Noting that the sensor placement problem in (Diwold, Ruhnke, and Middendorf 2010) evaluated a fixed number of sensors (five), and recalling that the Richmond, VA network has 872 nodes, the number of possible scenarios is in the order of 4×10^{12}. Since it is not feasible to search this decision space completely, a random selection of 2000 contamination events is chosen from a uniform distribution (this just means 2000 selections (with replacement?) of five nodes is sampled from the full decision space.)

In this example approach, the problem is formulated differently. As a result, we can only compare the outcome.

In [11]:
```
scenario_names = wn.junction_name_list
wn.options.solver.damplimit = 0.01
sim = wntr.sim.EpanetSimulator(wn)
sim.run_sim(save_hyd=True)
wn.options.quality.mode = 'TRACE'
```

Set up the Contamination Hydraulic Model

To be more computationally efficient, the contamination hydraulic model should be initialized before the sensor placement problem is run.

What this essentially produces is a unique hydraulic model for each scenario, which results in one contamination event at each feasible node for each flow pattern considered.

Simulations can are run for different scenarios of contamination *at each junction*, thereby covering the entire decision space. These contaminant tracer scenarios are then saved to disk and recalled later for the sensor placement problem.

In [12]:
```
# Run trace simulations (one from each junction) and extract data needed fo
r
# sensor placement optimization. You can run this step once, save the data
 to a
# file, and reload the file for sensor placement

# note this only has to be run ONCE
# before the sensor placement problem is executed
# IT TAKES A LONG TIME FOR A LARGE NETWORK

signal = pd.DataFrame()
# for inj_node in scenario_names:
#     wn.options.quality.trace_node = inj_node
#     wn.options.solver.damplimit = 0.1
#     sim_results = sim.run_sim(use_hyd=True)
#     trace = sim_results.node['quality']
#     trace = trace.stack()
#     trace = trace.reset_index()
#     trace.columns = ['T', 'Node', inj_node]
#     signal = signal.combine_first(trace)
# signal.to_csv('test_signal.csv')
```

Define the set of feasible sensor locations

Feasible sensor locations are determined by the physical location, sample times (1h timestep over 24h cycle), and the detection threshold (20 mg/L).

Then, for each contamination scenario at each node, extract the minimum detection time.

Define feasible sensors using location, sample times, and detection threshold

The hydraulic model characterizing the movement of contaminated water in each scenario runs for 24 hours in steps of 1 hour.

In [44]:
```
# check if the signal dataframe was generated in this run,
# or if it needs to be loaded from memory
if len(signal) == 0:
    signal = pd.read_csv('test_signal.csv')
```

In [15]:
```
sensor_names = wn.junction_name_list
# sample times starts at zero, ends at 86400 s (24 hours), and
# the hydraulic model proceeds in steps of 1h.
sample_times = np.arange(0, wn.options.time.duration, wn.options.time.hydra
ulic_timestep)
threshold = 20
sensors = {}
for location in sensor_names:
    position = chama.sensors.Stationary(location)
    detector = chama.sensors.Point(threshold, sample_times)
    stationary_pt_sensor = chama.sensors.Sensor(position, detector)
    sensors[location] = stationary_pt_sensor


# Extract all sensor-scenario detection times
det_times = chama.impact.extract_detection_times(signal, sensors)
det_time_stats = chama.impact.detection_time_stats(det_times)

# Extract minimum detection time for each scenario-sensor pair
min_det_time = det_time_stats[['Scenario','Sensor','Min']].copy()
min_det_time.rename(columns = {'Min':'Impact'}, inplace = True)
```

In [109]:
```
scenario_cov = chama.impact.detection_times_to_coverage(det_times, coverage
_type='scenario')
```

Run the Sensor Placement Optimization Algorithm

Note that the Chama package formulates the objective function in the form of the p-mean facility location problem, which is different than what is presented in (Diwold, Ruhnke, and Middendorf 2010). However, it is possible to substitute the methodology for a customized optimization algorithm.

In [28]:
```
#   Run sensor placement optimization to minimize detection time using 0 to
5 sensors
#   The impact for undetected scenarios is set at 1.5x the max sample time
#   Sensor cost is defined uniformly using a value of 1.  This means that
#   sensor_budget is equal to the number of sensors to place
impactform = chama.optimize.ImpactFormulation()
scenario_characteristics = pd.DataFrame({'Scenario': scenario_names,
                                         'Undetected Impact': sample_times.
max()*1.5})

sensor_characteristics = pd.DataFrame({'Sensor': sensor_names,'Cost': 1})
sensor_budget = [5]#[0,1,2,3,4,5]
impact_results = {}
for n in sensor_budget:
    impactform = chama.optimize.ImpactFormulation()
    coveragform = chama.optimize.CoverageFormulation()

    impact_results[n] = impactform.solve(min_det_time, sensor_characteristi
cs,
                                         scenario_characteristics, n)
```

In [118]:
```
coverage_formulation = chama.optimize.CoverageFormulation()
coverage_results = coverage_formulation.solve(scenario_cov, sensor_budget=5
)
```

Present Results of the Optimization Problem

Note that for the zero sensor case, the results suggest the detection occurs in 34.5 hours. This is a vestige of the model scenario duration, as all scenarios feature a maximum detection time of 34.5 hours. In many studies, a node reaching the maximum duration without detection is considered a non-detection, which is factored into the second optimization criterion concerning the detection rate.

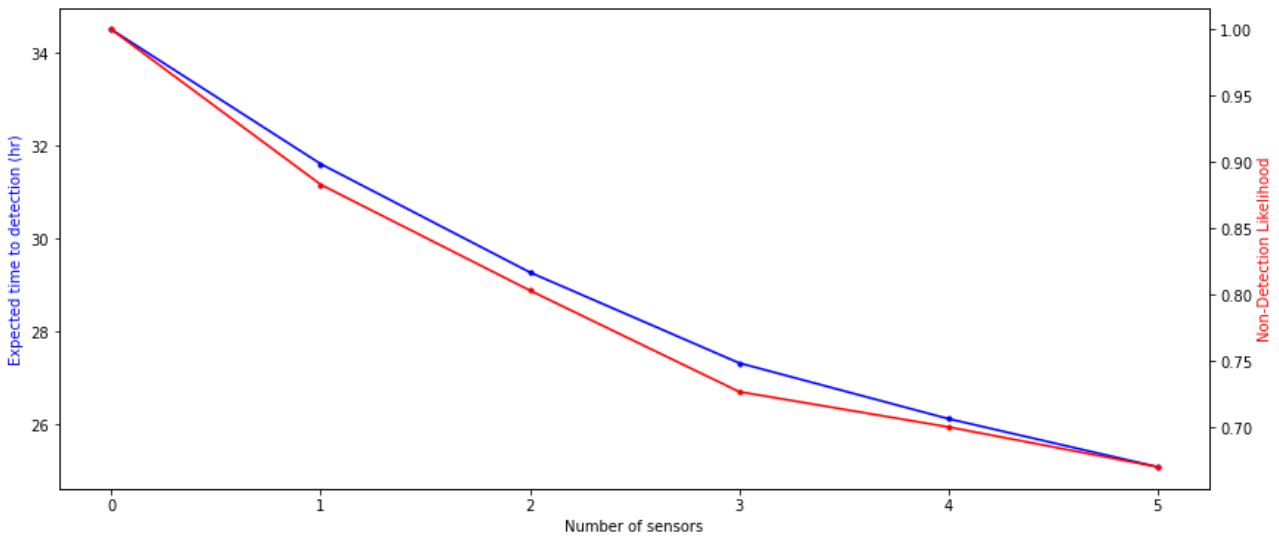
An interesting result is the marginal decrease in estimated time to detection for each additional sensor. Many studies translate the expected detection time to both volumetric consumption of contaminated water, and size of affected population, which translates the objective function into different terms that are perhaps more relevant to the true nature of the contamination problem.

In [126]:
```
# Plot objective for each sensor placement
objective_values = [results[n]['Objective']/3600 for n in sensor_budget]
detected_fractions = [results[n]['FractionDetected'] for n in results]

# convert coverage to non-detection likelihood
non_detect_likelihood = [1 - e for e in detected_fractions]

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(sensor_budget, objective_values, 'b', marker='.')
ax2.plot(sensor_budget, non_detect_likelihood, 'r', marker='.')
ax1.set_xlabel('Number of sensors')
ax1.set_ylabel('Expected time to detection (hr)', color='b')
ax2.set_ylabel('Non-Detection Likelihood', color='r')
```

Out[126]:
```
Text(0, 0.5, 'Non-Detection Likelihood')
```



Compare the Above Results to the Study Objectives and Results

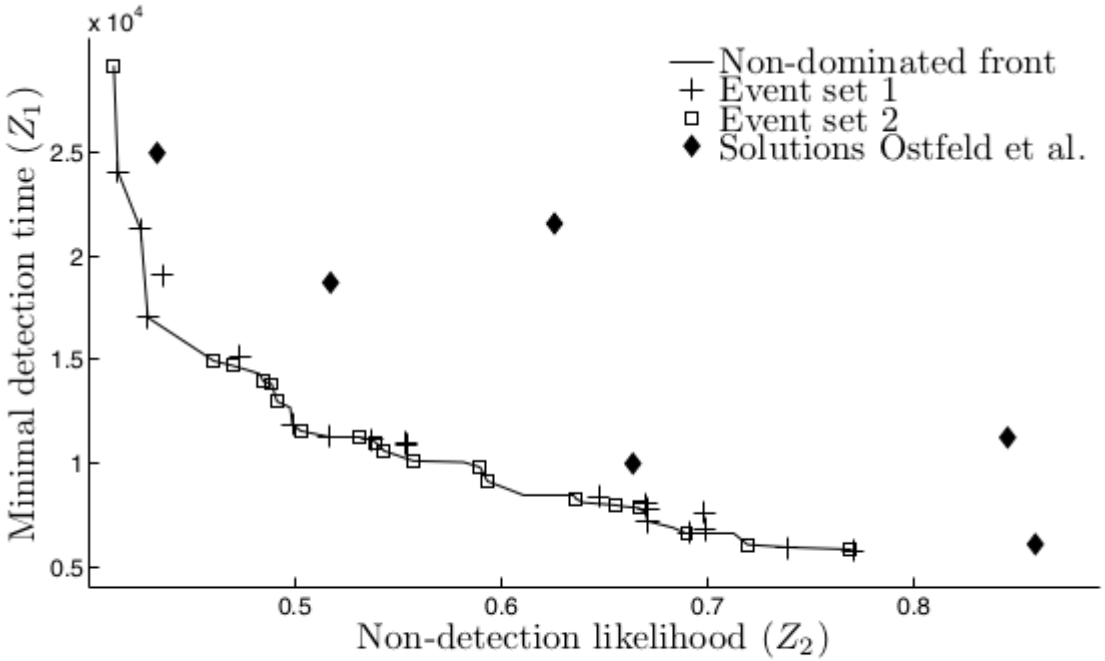
The default 'coverage' optimization problem in Chama finds a probabilistic lowest 'expected value' of the time to detection.

Compare the 'detection likelihood', above, and non-detection likelihood, below, we see that the Chama solution is in the vicinity of the Pareto front derived in the study, however the detection time metrics are different.

Question:

- What is a more realistic metric for the detection time?
  - As described in (Diwold, Ruhnke, and Middendorf 2010), Z_1 is *"the minimal contamination detection time of a sensor placement S on the **detectable** (emphasis mine) set of contaminations"*.
  - the default setting in the Chama package is 'expected time to detection'.

The 'cost' of an undetected event is set by default at 1.5x the simulation duration. What could we do to improve the interpretability of this metric? How could we change the representation? Is 'expected time to detection' any better?



(b) Network 2, $p = 5$, contamination scenario set 2

Above: results for the 5 sensor problem from (Diwold, Ruhnke, and Middendorf 2010).

Plot the Approximate Optimal Solution

Plot both the sensor placement layout representing the approximate solution, as well as a 'heat map' of the detection times for all nodes in the system.

In [127]:
```
# Plot selected sensors
n = 5
selected_sensors = results[n]['Sensors']
wntr.graphics.plot_network(wn, node_attribute=selected_sensors,
                           title='Selected sensors, n = {}'.format(n))

# Plot detection time for each scenario, when using some number of sensors
# assessment = results[n]['Assessment']
# assessment.set_index('Scenario', inplace=True)
# wntr.graphics.plot_network(wn, node_attribute=assessment['Impact']/3600,
#                            title='Detection time (hr), n = {}'.format(n))
```

Out[127]:
```
(<matplotlib.collections.PathCollection at 0x7f719b467fd0>,
 <matplotlib.collections.LineCollection at 0x7f719b22d5c0>)
```

Probability of Sensor Placement

An ant creates a solution iteratively, accounting for the sensors it has already placed.

Open Source Frameworks

The open-source Python package WNTR (Water Network Tool for Resilience) (https://wntr.readthedocs.io/en/latest/), developed by Sandia National Laboratories, is used here to demonstrate the setup and execution of a hydraulic model that implements an ACO algorithm similar to that presented in (Diwold, Ruhnke, and Middendorf 2010).

The WNTR package is extremely powerful and is equipped for analyzing many problems related to MWNs, including hydraulic modelling, and optimal sensor placement for such problems as leak localization and contamination detection.

For the contaminant detection problem, the WNTR package uses a mixed integer solver that is developed in the Chama (https://chama.readthedocs.io/en/latest/overview.html) Python library, which specifically addresses the optimal sensor placement problem. If you follow the link, you'll see how wide a range of applications this problem applies to, including oil and gas, process safety, water security, seismic and environmental monitoring, and asset protection and management.

References

- Diwold, Konrad, Thomas Ruhnke, and Martin Middendorf. 2010. “Sensor Placement in Water Networks Using a Population-Based Ant Colony Optimization Algorithm.” In, 6423:426–37. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Klise, K.A., Murray, R., Haxton, T. (2018). An overview of the Water Network Tool for Resilience (WNTR), In Proceedings of the 1st International WDSA/CCWI Joint Conference, Kingston, Ontario, Canada, July 23-25, 075, 8p.
- Klise, K.A., Bynum, M., Moriarty, D., Murray, R. (2017). A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study, Environmental Modelling and Software, 95, 420-431, doi: 10.1016/j.envsoft.2017.06.022
- Klise, K.A., Hart, D.B., Moriarty, D., Bynum, M., Murray, R., Burkhardt, J., Haxton, T. (2017). Water Network Tool for Resilience (WNTR) User Manual, U.S. Environmental Protection Agency Technical Report, EPA/600/R-17/264, 47p.
- Klise, K.A., Nicholson, B., and Laird, C.D. (2017). Sensor Placement Optimization using Chama, Sandia Report SAND2017-11472, Sandia National Laboratories.
- Ostfeld, Avi, James G. Uber, Elad Salomons, Jonathan W. Berry, William E. Hart, Cindy A. Phillips, Jean-Paul Watson, et al. 2008. “The Battle of the Water Sensor Networks (Bwsn): A Design Challenge for Engineers and Algorithms.” Journal of Water Resources Planning and Management 134 (6): 556–68.
- Preis, Ami, and Avi Ostfeld. 2008. “Multiobjective Contaminant Sensor Network Design for Water Distribution Systems.” Journal of Water Resources Planning and Management 134 (4): 366–77.