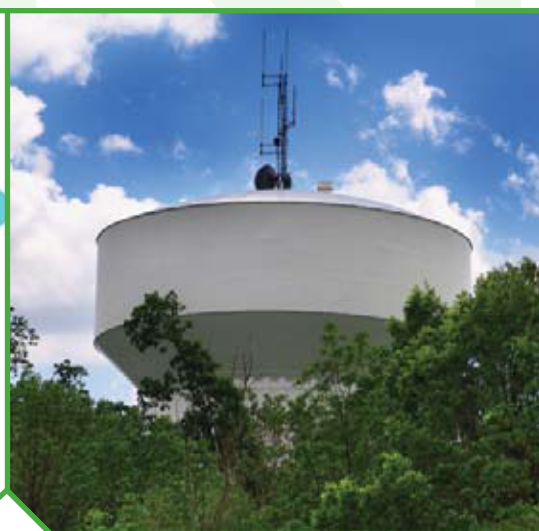
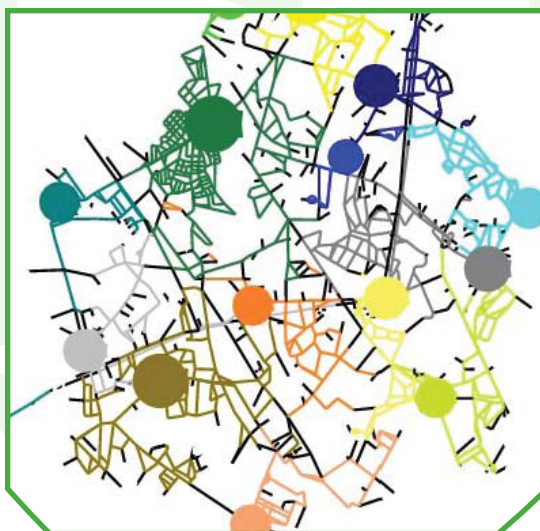


User's Manual TEVA-SPOT Toolkit

VERSION 2.5.2



USER'S MANUAL TEVA-SPOT TOOLKIT 2.5.2

by

Jonathan Berry, Erik Boman, Lee Ann Riesen
Scalable Algorithms Dept
Sandia National Laboratories
Albuquerque, NM 87185

William E. Hart, Cynthia A. Phillips, Jean-Paul Watson
Discrete Math and Complex Systems Dept
Sandia National Laboratories
Albuquerque, NM 87185

Project Officer:

REGAN MURRAY

NATIONAL HOMELAND SECURITY RESEARCH CENTER
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CINCINNATI, OH 45256

The U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and collaborated in the research described here under an Inter-Agency Agreement with the Department of Energy's Sandia National Laboratories (IAG # DW8992192801). This document has been subjected to the Agency's review, and has been approved for publication as an EPA document. EPA does not endorse the purchase or sale of any commercial products or services.

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:

Regan Murray
USEPA/NHSRC (NG 16)
26 W Martin Luther King Drive
Cincinnati OH 45268
(513) 569-7031
Murray.Regan@epa.gov



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.



Forward

Since its inception in 1970, EPA's mission has been to pursue a cleaner, healthier environment for the American people. The Agency was assigned the daunting task of repairing the damage already done to the natural environment and establishing new criteria to guide Americans in making a cleaner environment a reality. Since 1970, the EPA has worked with federal, state, tribal, and local partners to advance its mission to protect human health and the environment. In order to carry out its mission, EPA employs and collaborates with some of the nation's best scientific minds. EPA prides itself in applying sound science and state of the art techniques and methods to develop and test innovations that will protect both human health and the environment.

Under existing laws and recent Homeland Security Presidential Directives, EPA has been called upon to play a vital role in helping to secure the nation against foreign and domestic enemies. The National Homeland Security Research Center (NHSRC) was formed in 2002 to conduct research in support of EPA's role in homeland security. NHSRC research efforts focus on five areas: water infrastructure protection, threat and consequence assessment, decontamination and consequence management, response capability enhancement, and homeland security technology testing and evaluation. EPA is the lead federal agency for drinking water and wastewater systems and the NHSRC is working to reduce system vulnerabilities, prevent and prepare for terrorist attacks, minimize public health impacts and infrastructure damage, and enhance recovery efforts.

This Users Manual for the TEVA-SPOT Toolkit software package is published and made available by EPA's Office of Research and Development to assist the user community and to link researchers with their clients.

Jonathan Herrmann, Director

National Homeland Security Research Center
Office of Research and Development
U. S. Environmental Protection Agency

License Notice

TEVA-SPOT Toolkit is Copyright 2008 Sandia Corporation. Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation, the U.S. Government retains certain rights in this software.

The “library” refers to the TEVA-SPOT Toolkit software, both the executable and associated source code. This library is free software; you can redistribute it and/or modify it under the terms of the BSD License as published by the Free Software Foundation.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

The TEVA-SPOT Toolkit utilizes a variety of external executables that are distributed under separate open-source licenses:

- **PICO** - BSD and Common Public License
- **randomsample**, **sideconstraints** - ATT Software for noncommercial use.
- **ufl** - Common Public License

Acknowledgements

The National Homeland Security Research Center would like to acknowledge the following organizations and individuals for their support in the development of the TEVA-SPOT Toolkit User's Manual and/or in the development and testing of the TEVA-SPOT Toolkit Software.

Office of Research and Development - National Homeland Security Research Center

Robert Janke
Regan Murray
Terra Haxton

Sandia National Laboratories

Jonathan Berry
Erik Boman
William Hart
Lee Ann Riesen
Cynthia Phillips
Jean-Paul Watson
David Hart
Katherine Klise

Argonne National Laboratory

Thomas Taxon

University of Cincinnati

James Uber

American Water Works Association Utility Users Group

Kevin Morley

Contents

1	Introduction	1
1.1	What is TEVA-SPOT?	1
1.2	About This Manual	2
2	TEVA-SPOT Toolkit Basics	3
2.1	Approaches to Sensor Placement	3
2.2	The Main Steps in Using SPOT	4
2.2.1	Simulating Contamination Incidents	4
2.2.2	Computing Contamination Impacts	4
2.2.3	Performing Sensor Placement	5
2.2.4	Evaluating a Sensor Placement	5
2.3	Installation and Requirements for Using SPOT	6
2.4	Reporting Bugs and Feature Requests	7
3	Sensor Placement Formulations	8
3.1	The Standard SPOT Formulation	8
3.2	Robust SPOT Formulations	9
3.3	Min-Cost Formulations	10
3.4	Formulations with Multiple Objectives	10
3.5	The SPOT Formulation with Imperfect Sensors	11
4	Contamination Incidents and Impact Measures	12
4.1	Simulating Contamination Incidents	12
4.2	Using tso2Impact	12
4.3	Impact Measures	13
4.4	Advanced Tools for Large Sensor Placements Problems	15
5	Sensor Placement Solvers	16
5.1	A Simple Example	16
5.2	Computing a Bound on the Best Sensor Placement Value	20

5.3	Minimizing the Number of Sensors	20
5.4	Fixing Sensor Placement Locations	21
5.5	Robust Optimization of Sensor Locations	22
5.6	Multi-Criteria Analysis	23
5.7	Sensor Placements without Penalties	26
5.8	Limited-Memory Sensor Placement Techniques	28
5.9	Two-tiered Sensor Placement Approach	29
5.10	Evaluating a Sensor Placement	30
5.11	Sensor Placement with Imperfect Sensors	32
5.12	Summary of Solver Features	33
6	File Formats	35
6.1	TSG File	35
6.2	TSI File	36
6.3	DVF File	36
6.4	TAI File	37
6.5	ERD File	39
6.6	Sensor Placement File	39
6.7	Impact File	40
6.8	LAG File	40
6.9	Scenario File	41
6.10	Node File	41
6.11	Sensor Placement Configuration File	41
6.12	Sensor Placement Costs File	42
6.13	Placement Locations File	42
6.14	Sensor Class File	43
6.15	Junctions Class File	43
Appendix		
A	Unix Installation	47
A.1	Downloading	47
A.2	Configuring and Building	47

B	Data Requirements	48
C	Executable evalsensor	57
C.1	Overview	57
C.2	Command-Line Help	57
C.3	Description	58
D	Executable filter_impacts	59
D.1	Overview	59
D.2	Usage	59
D.3	Options	59
D.4	Arguments	59
D.5	Description	59
D.6	Notes	59
E	Executable PICO	60
E.1	Overview	60
E.2	Usage	60
E.3	Options	60
E.4	Description	60
E.5	Notes	60
F	Executable randomsample	61
F.1	Overview	61
F.2	Usage	61
F.3	Arguments	61
F.4	Description	61
F.5	Notes	61
G	Executable scenarioAggr	62
G.1	Overview	62
G.2	Usage	62
G.3	Options	62
G.4	Description	62

G.5	Notes	62
H	Executable createIPData	63
H.1	Overview	63
H.2	Command-Line Help	63
H.3	Description	63
H.4	Notes	63
I	Executable sideconstraints	64
I.1	Overview	64
I.2	Usage	64
I.3	Arguments	64
I.4	Description	64
J	Executable sp	65
J.1	Overview	65
J.2	Command-Line Help	65
J.3	Description	68
J.4	Notes	69
K	Executable sp-2tier	70
K.1	Overview	70
K.2	Command-Line Help	70
K.3	Description	71
K.4	Notes	72
L	Executable spotSkeleton	73
L.1	Overview	73
L.2	Usage	73
L.3	Description	73
L.4	Notes	73
M	Executable tevasim	75
M.1	Overview	75
M.2	Command-Line Help	75

N Executable tso2Impact	76
N.1 Overview	76
N.2 Command-Line Help	76
N.3 Description	77
N.4 Notes	77
O Executable ufl	78
O.1 Overview	78
O.2 Usage	78
O.3 Options	78
O.4 Arguments	78
O.5 Description	78

1 Introduction

Drinking water distribution systems are inherently vulnerable to accidental or intentional contamination because of their distributed geography. Further, there are many challenges to detecting contaminants in drinking water systems: municipal distribution systems are large, consisting of hundreds or thousands of miles of pipe; flow patterns are driven by time-varying demands placed on the system by customers; and distribution systems are looped, resulting in mixing and dilution of contaminants. The use of on-line, real-time contaminant warning systems (CWSs) is a promising strategy for mitigating these risks. Online sensor data can be combined with public health surveillance systems, physical security monitoring, customer complaint surveillance, and routine sampling programs to effect a rapid response to contamination incidents [22].

A variety of technical challenges need to be addressed to make CWSs a practical, reliable element of water security systems. A key aspect of CWS design is the strategic placement of sensors throughout the distribution network. Given a limited number of sensors, a desirable sensor placement minimizes the potential economic and public health impacts of a contaminant incident. There are a wide range of important design objectives for sensor placements (e.g., minimizing the cost of sensor installation and maintenance, the response time to a contamination incident, and the extent of contamination). In addition, flexible sensor placement tools are needed to analyze CWS designs in large scale networks.

1.1 What is TEVA-SPOT?

The Threat Ensemble Vulnerability Assessment and Sensor Placement Optimization Tool (TEVA-SPOT) has been developed by the U. S. Environmental Protection Agency, Sandia National Laboratories, Argonne National Laboratory, and the University of Cincinnati. TEVA-SPOT has been used to develop sensor network designs for several large water utilities [12], including the pilot study for EPA’s Water Security Initiative.

TEVA-SPOT allows a user to specify a wide range of modeling inputs and performance objectives for contamination warning system design. Further, TEVA-SPOT supports a flexible decision framework for sensor placement that involves two major steps: a modeling process and a decision-making process [13]. The modeling process includes (1) describing sensor characteristics, (2) defining the design basis threat, (3) selecting impact measures for the CWS, (4) planning utility response to sensor detection, and (5) identifying feasible sensor locations.

The design basis threat for a CWS is the ensemble of contamination incidents that a CWS should be designed to protect against. In the simplest case, a design basis threat is a contamination scenario with a single contaminant that is introduced at a specific time and place. Thus, a design basis threat consists of a set of contamination incidents that can be simulated with standard water distribution system modeling software [18]. TEVA-SPOT provides a convenient interface for defining and computing the impacts of design basis threats. In particular, TEVA-SPOT can simulate many contamination incidents in parallel, which has reduced the computation of very large design basis threats from weeks to hours on EPA’s high performance computing system.

TEVA-SPOT was designed to model a wide range of sensor placement problems. For example, TEVA-SPOT supports a number of impact measures, including the number of people exposed to dangerous levels of a contaminant, the volume of contaminated water used by customers, the number of feet of contaminated pipe, and the time to detection. Response delays can also be specified to account for the time a water utility would need to verify a contamination incident before notifying the public. Finally, the user can specify the feasible locations for sensors and fix sensor locations during optimization. This flexibility allows a user to evaluate how different factors impact the CWS performance and to iteratively refine a CWS design.

1.2 About This Manual

The capabilities of TEVA-SPOT can be accessed either with a GUI or from command-line tools. This user manual describes the TEVA-SPOT Toolkit, which contains these command-line tools. The TEVA-SPOT Toolkit can be used within either a MS Windows DOS shell or any standard Unix shell (e.g. the Bash shell).

The following sections describe the TEVA-SPOT Toolkit, which is referred to as SPOT throughout this manual:

- **TEVA-SPOT Toolkit Basics** - An introduction to the process of sensor placement, the use of SPOT command-line tools, and installation of the SPOT executables.
- **Sensor Placement Formulations** - The mathematical formulations used by the SPOT solvers.
- **Contamination Incidents and Impact Measures** - A description of how contamination incidents are computed, and the impact measures that can be used in SPOT to analyze them.
- **Sensor Placement Solvers** - A description of how to apply the SPOT sensor placement solvers.
- **File Formats** - Descriptions of the formats of files used by the SPOT solvers.

In addition, the appendices of this manual describe the syntax and usage of the SPOT command-line executables.

2 TEVA-SPOT Toolkit Basics

This section provides an introduction to the process of sensor placement, the use of SPOT command-line tools, and the installation of the SPOT executables.

2.1 Approaches to Sensor Placement

Sensor placement strategies can be broadly characterized by the technical approach and the type of computational model used. For a review of sensor placement methods, see Hart and Murray, JWRPM November 2010. The following categories reflect important differences in proposed sensor placement strategies:

- **Expert Opinion:** Although expertise with water distribution systems is always needed to design an effective CWS, some approaches are solely guided by expert judgment. For example, Berry et al. [4] and Trachman [21] consider sensor placements developed by experts with significant knowledge of water distribution systems. These experts did not use computational models to carefully analyze network dynamics. Instead, they used their experience to identify locations whose water quality is representative of water throughout the network.
- **Ranking Methods:** A related approach is to use preference information to rank network locations [1, 8]. In this approach, a user provides preference values for the properties of a "desirable" sensor location, such as proximity to critical facilities. These preferences can then be used to rank the desirability of sensor locations throughout the network. Further, spatial information can be integrated to ensure good coverage of the network.
- **Optimization:** Sensor placement can be automated with optimization methods that computationally search for a sensor configuration that minimizes contamination risks. Optimization methods use a computational model to estimate the performance of a sensor configuration. For example, a model might compute the expected impact of an ensemble of contamination incidents, given sensors placed at strategic locations.

Optimization methods can be further distinguished by the type of computational model that they use. Early sensor placement research focused on models that used simplified network models derived from contaminant transport simulations. For example, hydraulic simulations can be used to model stable network flows [3], or to generate an *averaged* water network flow model [15].

More recently, researchers have used models that directly rely on contaminant transport simulation results. Simulation tools, like EPANET [18], perform 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 they can model the transport of contaminants and related chemical interactions. Thus, the CWS design process can directly minimize contamination risks by considering simulations of an ensemble of contamination incidents, which reflect the impact of contamination at different locations, times of the day, etc.

SPOT development has focused on optimization methods, and in particular on methods that use contaminant transport simulation. Contaminant transport simulation models can directly model contamination risks, and consequently optimization methods using these models have proven effective at minimizing risk. Comparisons with expert opinion and ranking methods suggest that these approaches are not as effective in large, complex networks [4, 16]. Further, optimization methods using simpler models can fail to capture important transient dynamics (see Berry et al. [6] for a comparison).

A key issue for the simulation-based optimization methods is that they require the simulation of a potentially large number of contamination incidents. Consequently, it is very expensive to apply generic optimization methods like evolutionary algorithms [15]. However, Berry et al. [5] have shown that these simulations can

be performed in an off-line preprocessing step that is done in advance of the optimization process. Thus, the time needed for simulation does not necessarily impact the time spent performing sensor placement.

2.2 The Main Steps in Using SPOT

The following example illustrates the main steps required to (1) simulate contamination incidents, (2) compute contamination impacts, (3) perform sensor placement, and (4) evaluate a sensor placement. This example places sensors in EPANET Example 3 (Net3), a small distribution system with 97 junctions.

The data files used in this example are available with the SPOT software package (C:\spot\examples\simple directory). Installation instructions are included in **Section 2.3** (p. 6). In general, a user will need to use a variety of data sources to develop a sensor placement model. Data requirements for TEVA-SPOT are described in detail in the **Appendix** (p. 48). File formats are described in **Section 6** (p. 35).

2.2.1 Simulating Contamination Incidents

Simulation of contamination incidents is performed with the `tevasim` command, which iteratively calls EPANET to simulate an ensemble of contamination incidents. The `tevasim` command has the following inputs and outputs:

- **Inputs:**

- TSG File: defines an ensemble of contamination scenarios
- INP File: the EPANET input file for the network

- **Outputs:**

- ERD File: a binary file that stores the contamination results for all incidents (ERD database files include a `erd`, `index.erd`, `hyd.erd`, and `qual.erd` file)
- OUT File: a plain text log file

For example, the file C:\spot\examples\simple\Net3.tsg defines an ensemble of contamination scenarios for Net3. Contamination incidents are simulated for all network junctions, one for each hour of the day, and each contamination incident models an injection that continues for 24 hours. The `tevasim` command performs these contaminant transport simulations, using the following command line:

```
tevasim --tsg Net3.tsg Net3.inp Net3.out Net3
```

2.2.2 Computing Contamination Impacts

TSO and ERD files contain raw data about the extent of a contamination throughout a network. This data needs to be post-processed to compute relevant impact statistics. The `tso2Impact` command processes a TSO and ERD files and generates one or more IMPACT files. An IMPACT file is a plain text file that summarizes the consequence of each contamination incident in a manner that facilitates optimization. The `tso2Impact` command has the following inputs and outputs:

- **Inputs:**

- ERD or TSO File: a binary file that stores the contamination result data generated by `tevasim`

- **Outputs:**

- IMPACT File(s): plain text files that summarize the observed impact at each location where a contamination incident could be observed by a potential sensor.

- NODMAP File(s): plain text files that map sensor placement ids to the network junction labels (defined by EPANET).

The **tso2Impact** command generates IMPACT files with the following command line:

```
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.erd
```

This command generates IMPACT files for each of the five objectives specified: mass consumed (mc), volume consumed (vc), time to detection (td), number of failed detections (nfd) and extent of contamination (ec). For each impact file (e.g. **Net3_mc.impact**), a corresponding id file is generated (e.g. **Net3_mc.impact.id**).

The ERD file format replaced the TSO and SDX index file format, created by previous versions of **tevasim**, to extend **tevasim** capability for multi-specie simulation using EPANET-MSX. While **tevasim** produces only ERD files (even for single specie simulation), **tso2Impact** accepts both ERD and TSO file formats.

2.2.3 Performing Sensor Placement

An IMPACT file can be used to define a sensor placement optimization problem. The standard problem supported by SPOT is to minimize the expected impact over an ensemble of incidents while limiting the number of potential sensors. By default, sensors can be placed at any junction in the network. The **sp** command coordinates the application of optimization solvers for sensor placement. The **sp** command has a rich interface, but the simplest use of it requires the following inputs and outputs:

- **Inputs:**

- IMPACT File(s): plain text files that summarize the observed impact at each location
- NODMAP File(s): plain text files that map sensor placement ids to the network junction labels

- **Outputs:**

- SENSORS File: a plain text file that summarizes the sensor locations identified by the optimizer

For example, the command

```
sp --path=$bin --print-log --network="Net3" --objective=mc --solver=snl_grasp \
  --ub=ns,5 --seed=1234567
```

generates the file **Net3.sensors**, and prints a summary of the impacts for this sensor placement.

2.2.4 Evaluating a Sensor Placement

The final output provided by the **sp** command is actually generated by the **evalsensor** command, and this command can be directly used to evaluate a sensor placement for a wide variety of different objectives. The **evalsensor** command requires the following inputs:

- **Inputs:**

-
- IMPACT File(s): plain text files that summarize the observed impact at each location
- NODMAP File(s): plain text files that map sensor placement ids to the network junction labels
- SENSORS File: a plain text file that defines a sensor placement

For example, the command

```
evalsensor --nodemap=Net3.nodemap Net3.sensors Net3_ec.impact Net3_mc.impact \
Net3_nfd.impact
```

will summarize the solution in the `Net3.sensors` file for the `ec`, `mc` and `nfd` impact measures. No files are generated by `evalsensors`.

2.3 Installation and Requirements for Using SPOT

Instructions for installing SPOT in Unix are included in the **Appendix** (p. 47).

To install SPOT on MS Windows platforms, an installer executable can be downloaded from

<https://software.sandia.gov/trac/spot/downloader>

Note that the first time this page is accessed, the user must register. When run, this installer places the SPOT software in the directory

`C:\tevaspot`

The installer places executables in the directory

`C:\tevaspot\bin`

This directory should be added to the system PATH environment variable to allow SPOT commands to be run within any DOS shell.

Some of the SPOT commands use the Python scripting language. Python is not commonly installed in MS Windows machines, but an installer script can be downloaded from

<http://www.python.org/download/>

The system path needs to be modified to include the Python executable. A nice video describing how to edit the system path is available at:

<http://showmedo.com/videos/video?name=960000&fromSeriesID=96>

No other utilities need to be installed to run the SPOT commands. EPANET is linked into the `tevasim` executable. Detailed information about the SPOT commands is provided on the SPOT wiki:

<https://software.sandia.gov/trac/spot/wiki/Tools>

Note that all SPOT commands need to be run from the DOS command shell. This can be launched from the "Accessories/Command Prompt" menu. Numerous online tutorials can provide information about DOS commands. For example, see http://en.wikipedia.org/wiki/List_of_DOS_commands or <http://www.computerhope.com/msdos.htm>

The plain text input files used by SPOT can be edited using standard text editors. For example, at a DOS prompt you can type `notepad Net3.tsg` to open up the `Net3.tsg` file with the MS Windows Notepad application. The plain text output files can be viewed in a similar manner. The binary files generated by SPOT cannot be viewed in this manner. Generally, output files should not be modified manually since many are used as input to other programs.

2.4 Reporting Bugs and Feature Requests

The TEVA-SPOT development team uses Trac tickets to communicate requests for features and bug fixes. The TEVA-SPOT Trac site can be accessed at: <https://software.sandia.gov/trac/spot>. External users can insert a ticket, which will be moderated by the developers. Note that this is the only mechanism for ensuring that bug fixes will be made a high priority by the development team.

3 Sensor Placement Formulations

SPOT integrates solvers for sensor placement that have been developed by Sandia National Laboratories and the Environmental Protection Agency, along with a variety of academic collaborators [3, 5, 7, 9, 13, 14]. SPOT includes (1) general-purpose heuristic solvers that consistently locate optimal solutions in minutes, (2) integer- and linear-programming heuristics that find solutions of provable quality, (3) exact solvers that find globally optimal solutions, and (4) bounding techniques that can evaluate solution optimality. These solvers optimize a representation of the sensor placement problem that may be either implicit or explicit. However, in either case the mathematical formulation for this problem can be described.

This section describes the mixed integer programming (MIP) formulations optimized by the SPOT solvers. This presentation assumes that the reader is familiar with MIP modeling. First, the standard SPOT formulation, eSP, is described which minimizes expected impact given a sensor budget. Subsequently, several other sensor placement formulations that SPOT solvers can optimize are presented. This discussion is limited to a description of the mathematical structure of these sensor placement problems. In many cases, SPOT has more than one optimizer for these formulations. These optimizers are described later in this manual. However, the goal of this section is to describe the mathematical structure of these formulations.

3.1 The Standard SPOT Formulation

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 in SPOT, since it can be effectively used to select sensor placements in large water distribution networks.

A MIP formulation for expected-impact sensor placement is:

$$\begin{aligned}
 \text{(eSP)} \quad & \min \quad \sum_{a \in \mathcal{A}} \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} \\
 \text{s.t.} \quad & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 && \forall a \in \mathcal{A} \\
 & x_{ai} \leq s_i && \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\
 & \sum_{i \in L} c_i s_i \leq p \\
 & s_i \in \{0, 1\} && \forall i \in L \\
 & 0 \leq x_{ai} \leq 1 && \forall a \in \mathcal{A}, i \in \mathcal{L}_a
 \end{aligned}$$

This MIP minimizes the expected impact of a set of contamination incidents defined by \mathcal{A} . For each incident $a \in \mathcal{A}$, α_a is the weight of incident a , frequently a probability. This formulation integrates contamination simulation results, which are reported at a set of *locations* from the full set, denoted L , where a location refers to a network junction. For each incident a , $\mathcal{L}_a \subseteq L$ is the set of locations that can be contaminated by a . Thus, a perfect 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 $a \in \mathcal{A}$ 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 measure 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_{ia} variables indicate whether incident a is witnessed by a sensor at location i . A given set of sensors may not be able to witness all contamination incidents. To account for this, L contains a *dummy* location, q . This dummy location is in all subsets \mathcal{L}_a . If the dummy location witnesses and incident, it generally means that no real sensor can detect that incident. The impact for this location is handled in two different ways: (1) it 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 CWS, or (2) it has zero impact. The first approach treats detection by this dummy location as a penalty. The second approach simply ignores the detection by this dummy, though this only makes sense with an additional side-constraint on the maximum number of failed detections. Without this extra side constraint, selecting the dummy to witness each event, essentially placing no sensors, would appear to give zero impact, or total protection.

The current implementation of eSP contains an additional set of constraints for the case where dummy impact is zero:

$$x_{aq} \leq 1 - s_i \quad \forall a \in \mathcal{A}, i \in \mathcal{L}_a \setminus \{q\}.$$

This constraint does not allow the selection of the zero-impact dummy for incidents that are truly witnessed by a real sensor. Because this constraint only makes sense when coupled with a side constraint on the maximum number of failed detections, we will not explicitly include it in MIP formulations in this section. However, the user should be aware of its existence because it affects the computation of sensor placement average impact. Without this constraint, if a sensor placement is allowed to miss r incidents, then the MIP will choose the (zero-impact) dummy for the r highest impact events, whether the event is detected or not. With this constraint, all true detections are counted. Different sensor placements will have varying numbers of incidents contributing to the average impact. Thus the optimal sensor placement, and the value of the optimal sensor placement, will be different when this constraint is there compared to when it is not.

The eSP formulation with the above extra constraint set is a slight generalization of the sensor placement model described by Berry et al. [5]. Berry et al. treat the impact of the dummy as a penalty. In that case the extra constraints are redundant. The impact of a dummy detection is no smaller 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.

Berry et al. note that eSP without the extra constraints is identical to the well-known p -median facility location problem [11] 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.

3.2 Robust SPOT Formulations

The eSP model can be viewed as optimizing one particular statistic of the distribution of impacts defined by the contaminant transport simulations. However, other statistics may provide more "robust" solutions, that are less sensitive to changes in this distribution [24]. Consider the following reformulation of eSP:

$$\begin{aligned}
(\text{rSP}) \quad & \min \quad \text{Impact}_f(\alpha, d, x) \\
& \text{s.t.} \quad \sum_{i \in \mathcal{L}_a} x_{ai} = 1 \quad \forall a \in \mathcal{A} \\
& \quad \quad x_{ai} \leq s_i \quad \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\
& \quad \quad \sum_{i \in \mathcal{L}} c_i s_i \leq p \\
& \quad \quad s_i \in \{0, 1\} \quad \forall i \in \mathcal{L} \\
& \quad \quad 0 \leq x_{ai} \leq 1 \quad \forall a \in \mathcal{A}, i \in \mathcal{L}_a
\end{aligned}$$

The function $\text{Impact}_f(\alpha, d, x)$ computes a statistic of the impact distribution. The following functions are supported in SPOT (see Watson, Hart and Murray [24] for further discussion of these statistics):

- **Mean:** This is the statistic used in eSP.
- **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 [20]. The value of VaR is less than the TCE value

(see below).

Mathematically, suppose a random variable W describes the distribution of possible impacts. Then

$$\text{VaR}(W, \beta) = \min\{w \mid \Pr[W \leq w] \geq \beta\}.$$

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 which 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 with probability β . This value is between VaR and the worst-case value.

Mathematically, then

$$\text{TCE}(\beta) = \mathbb{E}[W \mid W \geq \text{VaR}(\beta)].$$

The Conditional Value-at-Risk (CVaR) is a linearization of TCE investigated by Uryasev and Rockafellar [17]. CVaR approximates TCE with a continuous, piecewise-linear function of β , which enables the use of CVaR in a MIP models for rSP.

- **Worst:** The worst impact value can be easily computed, since a finite number of contamination incidents are simulated. Further, rSP can be reworked to formulate a worst-case MIP formulation. However, this statistic is sensitive to changes in the number of contamination incidents that are modeled; adding additional contamination incidents may significantly impact this statistic.

3.3 Min-Cost Formulations

A standard variant of eSP and rSP is to minimize cost while constraining the impact to be below a specified threshold, u . For example, the eSP MIP can be revised to formulate a MIP to minimize cost:

$$\begin{aligned} (\text{ceSP}) \quad & \min \quad \sum_{i \in L} c_i s_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 & \forall a \in \mathcal{A} \\ & x_{ai} \leq s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\ & \sum_{a \in \mathcal{A}} \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} \leq u \\ & s_i \in \{0, 1\} & \forall i \in L \\ & 0 \leq x_{ai} \leq 1 & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \end{aligned}$$

Minimal cost variants of rSP can also be easily formulated.

3.4 Formulations with Multiple Objectives

CWS design generally requires the evaluation and optimization of a variety of performance objectives. Some performance objectives cannot be simultaneously optimized, and thus a CWS design must be selected from a trade-off between these objectives [23].

SPOT supports the analysis of these trade-offs with the specification of additional constraints on impact measures. For example, a user can minimize the expected extent of contamination (ec) while constraining the worst-case time to detection (td). SPOT allows for the specification of more than one impact constraint. However, the SPOT solvers cannot reliably optimize formulations with more than one impact constraint.

3.5 The SPOT Formulation with Imperfect Sensors

The previous sensor placement formulations make the implicit assumption that sensors work perfectly. That is, they never fail to detect a contaminant when it exists, and they never generate an erroneous detection when no contaminant exists. In practice, sensors are imperfect, and they generate these types of errors.

SPOT addresses this issue by supporting a formulation that models simple sensor failures [2]. Each sensor, s_i , has an associated probability of failure, p_i . With these probabilities, we can easily assess the probability that a contamination incident will be detected by a particular sensor. Thus, it is straightforward to compute the expected impact of a contamination incident.

This formulation does not explicitly allow for the specification of probabilities of false detections. These probabilities do not impact the performance of a CWS during a contamination incident. Instead, they impact the day-to-day maintenance and use of the CWS; erroneous detections create work for the CWS users, which is an ongoing cost. The overall likelihood of false detections is simply a function of the sensors that are selected. In cases where every sensor has the same likelihoods, this implies a simple constraint on the number of sensors.

4 Contamination Incidents and Impact Measures

This section describes how to simulate contamination incidents and compute contamination impacts, which are the first steps needed to setup and solve a sensor placement problem with SPOT. These two steps can be viewed as preprocessing or data preparation for sensor placement optimization. Thus, these steps can be performed prior to optimization, which is generally a more interactive, iterative process.

The following sections illustrate the capabilities of SPOT with the example in the C:\spot\examples\simple directory.

4.1 Simulating Contamination Incidents

To simulate contamination incidents, the **tevasim** (p. 75) command is utilized, which uses EPANET to perform an ensemble of contaminant transport simulations defined by a **TSG File** (p. 35). An ensemble of contamination scenarios for EPANET Example Net3 is defined in the file C:\spot\examples\simple\Net3.tsg. Contamination incidents are simulated for all network junctions, one for each hour of the day, and each contamination incident models an injection that continues for 24 hours. The **tevasim** command is run with the following command line:

```
tevasim --tsg Net3.tsg Net3.inp Net3.out Net3
```

This command generates several files: a binary ERD database that contains the contamination transport data (the database is stored in Net3.erd, Net3.index.erd, Net3-1.hyd.erd, and Net3-1.qual.erd), and Net3.out, which provides a textual summary of the EPANET simulations and is the same as the report file (*.rpt) from EPANET.

Adding **DVF File** (p. 36) for flushing control allows the user to simulate a flushing response to an event. **tevasim** behaves slightly differently in this case, adding in the flushing rate to demands at the specified nodes, and closing the specified pipes. The number of pipes and/or nodes can be zero, if you don't want to use that part of the response policy.

To simulate contamination incidents using multi-species reactions, **tevasim** uses the EPANET multi-species extension (MSX). EPANET-MSX is an extension to EPANET that enables complex reaction schemes between multiple chemical and biological species in both the bulk flow and at the pipe wall. For a review of EPANET-MSX, see Shang, Uber and Rossman [19]. Multi-species contamination incidents require a msx file and species declaration. The following **tevasim** command simulates multi-species contamination incidents:

```
tevasim --tsg Net3_bio.tsg --msx bio.msx --mss BIO Net3.inp Net3.out Net3_bio
```

4.2 Using tso2Impact

After running **tevasim** (p. 75) command, the output database, Net3.erd, can be used to compute one or more IMPACT files. An IMPACT file summarizes the consequence of each contamination incident in a manner that facilitates optimization. The **tso2Impact** (p. 76) command generates these files with the following command line:

```
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.erd
```

This command generates IMPACT files for each of the five objectives specified: mass consumed (mc), volume consumed (vc), time to detection (td), number of failed detections (nfd) and extent of contamination (ec).

For each IMPACT file (e.g. `Net3_mc.impact`), a corresponding ID file is generated to map the sensor placement ids back to the network junction labels (e.g. `Net3_mc.impact.id`).

The impact measures computed by `tso2Impact` represent the amount of impact that would occur up until the point where a contamination incident is detected. This computation assumes that sensors work perfectly (i.e., there are no false positive or false negative errors). However, the sensor behavior can be generalized in two ways. First, a detection threshold can be specified; contaminants are only detected above a specified concentration limit (the default limit is zero). Second, a response time can be specified, which accounts for the time needed to verify the presence of contamination (e.g. by field investigation) and then inform the public (the default response time is zero). The contamination impact is computed at the time where the response has completed (the detection time plus response time), which is called the *effective response time*. For undetected incidents, the effective response time is simply the end of the contaminant transport simulation. The following illustrates how to specify these options:

```
tso2Impact --responseTime 60 --detectionLimit 0.1 --mc Net3 Net3.erd
```

This computes impacts for a 60 minute response time, with a 0.1 detection threshold. Note that the units for `--detectionLimit` are the same as for the MASS values that are specified in the TSG file.

Impacts from multiple ERD files can be combined to generate a single IMPACT file using the following syntax:

```
tso2Impact --detectionLimit 30000000 --detectionLimit 0.0001 --mc Net3 Net3_1a.erd
Net3_1b.erd
```

Note that the value of 30000000 corresponds to the detection threshold for the contaminant described in `Net3_1a.erd` and 0.0001 is the detection threshold for the contaminant described in `Net3_1b.erd`. For example, this can be used to combine simulation results from different types of contaminants, in which the ERD files would have been generated from different TSG files. Murray et al. [13] use this technique to combine data from different types of contamination incidents into a single impact metric.

The `dvf` option specifies that the demands added due to flushing be subtracted out prior to calculating the impact measures. Add this in if the demands created by simulating flushing are not “consumed” - i.e., you don’t want the mass included in mass consumed, or other impacts. The time-extent-of-contamination (`tec`) impact measure integrates the time that each pipe contains contaminated water, rather than just the length of pipe that ever contains contaminated water. The result is in units of ft-hrs.

The `species` option specifies which species to use to compute impacts. This option is required for multi-species contamination incidents created by `tevasim`. For example:

```
tso2Impact --mc --vc --td --nfd --ec --species BIO Net3_bio Net3_bio.erd
```

4.3 Impact Measures

After running `tevasim` (p. 75) command, the output database, `Net3.erd`, can be used to compute one or more IMPACT files. An IMPACT file summarizes the consequence of each contamination incident in a manner that facilitates optimization. A variety of objective measures are supported by `tso2Impact` to reflect the different criteria that decision makers could use in CWS design. For most of these criteria, there is a *detected* and *undetected* version of the objective. This difference concerns how undetected contamination incidents are modeled.

For example, the default time-to-detection objective, `td`, uses the time at which the EPANET simulations are terminated to define the time for incidents that are not detected. The effect of this is that undetected incidents severely penalize sensor network designs. By contrast, the *detected* time-to-detection, `dtd`, simply

ignores these incidents (they have impact zero and do not affect the sensor network design). Sensor placement with the *detected* objective is somewhat more precise but optimization can be slow. Ideally, the objective is optimized with a revised formulation that explicitly limits the fraction of incidents that are not detected by the sensors. However, in real world applications, the detected metric is typically used without side constraints.

The following objectives are currently supported by **tso2Impact**:

- **ec** and **dec** - The extent of contaminated in the network. This is the total feet of pipes contaminated by the effective response time. An entire pipe is considered contaminated if contaminant enters the pipe at a given time step. For **ec**, the extent of contamination of an undetected incident is the extent of contamination at the point when the simulation terminates, while undetected contamination incidents are ignored for **dec**.
- **mc** and **dmc** - The mass of contaminant consumed by junctions in the network with nonzero demand. For **mc**, the mass of contaminant of an undetected incident is the mass of contaminant that has left the network via demand at the point when the simulation terminates, while undetected contamination incidents are ignored for **dmc**. This objective is typically measured in milligrams (the units used in the TSG file are mg/L). However, concentrations may also be interpreted; for example, we can treat this measure as a count of cells for a biological contaminant, where the TSG measurement is cells/L.
- **nfd** - The number of contamination incidents that are not detected by any sensor before the contaminant transport simulations terminate. NOTE: this measure is not affected by the response time option.
- **pe** and **dpe** - The number of individuals exposed to a contaminant. For **pe**, the population exposed for an undetected incident is the population exposed at the point when the simulation terminates, while undetected contamination incidents are ignored for **dpe**.
- **pd** and **dpd** - The number of individuals that receive a dose of contaminant above a specified threshold. For **pd**, the population dosed by an undetected incident is the population dosed at the point when the simulation terminates, while for **dpd** the undetected contamination incidents are ignored.
- **pk** and **dpk** - The number of individuals killed by a contaminant. For **pk**, the population killed by an undetected incident is the population killed at the point when the simulation terminates, while for **dpk** the undetected contamination incidents are ignored.
- **td** and **dttd** - The time, in minutes, from the beginning of a contamination incident until the first sensor detects it. For **td**, the time-to-detection of an undetected incident is the time from the start of the incident until the end of the simulation, while undetected contamination incidents are ignored for **dttd**. NOTE: this measure is not affected by the response time option.
- **vc** and **dvc** - The volume of contaminated water consumed by junctions in the network with nonzero demand. For **vc**, the volume of contaminated water of an undetected incident is the volume of contaminated water consumed at the point when the simulation terminates, while undetected contamination incidents are ignored for **dvc**.

These health impact measures are computed with an auxiliary input file, **TAI**, that specifies parameters for a health impact model that predicts how a population is affected by exposure to a contaminant. The **TAI File** (p. 37) **bio.tai** specifies the nature of the contaminant and how it impacts human health. Further, this file specifies the fraction of the volume of water consumed at junctions that is consumed by humans. For example, consider the command line:

```
tso2Impact --pe Net3 Net3.erd bio.tai
```

4.4 Advanced Tools for Large Sensor Placements Problems

In some applications, the size of the IMPACT files is very large, which can lead to optimization models that cannot be solved on standard 32-bit workstations. SPOT includes several utilities that are not commonly used to address this challenge: the **scenarioAggr** (p. 62) executable aggregates similar contamination incidents, and the **filter_impacts** (p. 59) script filters out contamination incidents that have low impacts.

The **scenarioAggr** (p. 62) executable reads an IMPACT file, finds similar incidents, combines them, and writes out another IMPACT file. This aggregation technique combines two incidents that impact the same locations in the same order, allowing for the possibility that one incident continues to impact other locations. For example, two contamination incidents should travel in the same pattern if they differ only in the nature of the contaminant, though one may decay more quickly than the other. Aggregated incidents can be combined by simply averaging the impacts that they observe and updating the corresponding incident weight.

For example, consider the command:

```
scenarioAggr --numEvents=236 Net3_mc.impact
```

This creates the files **aggrNet3_mc.impact** and **aggrNet3_mc.impact.prob**; where the **Net3_mc.impact** file has 236 events. The file **aggrNet3_mc.impact** is the new IMPACT file, and the file **aggrNet3_mc.impact.prob** contains the probabilities of the aggregated incidents.

The **filter_impacts** (p. 59) script reads an impact file, filters out the low-impact incidents, rescales the impact values, and writes out another IMPACT file. The command:

```
filter_impacts --percent=5 Net3_mc.impact filtered.impact
```

generates an IMPACT file that contains the incidents whose impacts (without sensors) are the largest 5% of the incidents in **Net3_mc.impact**. Similarly, the **--num=k** option selects the k incidents with the largest impacts, and the option **--threshold=h** selects the incidents with the impacts greater than or equal to h .

The **filter_impacts** command also includes options to rescale the impact values. The **--rescale** option rescales impact values with a log-scale and the **--round** option rescales impact values to rounded log-scale values.

5 Sensor Placement Solvers

The SPOT sensor placement solvers are launched with the **sp** (p. 65) command. The **sp** command reads in one or more IMPACT files, and computes a sensor placement. Command-line options for **sp** can specify any of a set of performance or cost goals as the objective to be optimized, as well as constraints on performance and cost goals.

The **sp** command currently interfaces with three different sensor placement optimizers:

- **MIP solvers** - Several different MIP solvers can be used by the **sp** command: the commercial CPLEX solver and the open-source PICO solver. These optimizers use the MIP formulations to find globally optimal solutions. However, this may be a computationally expensive process (especially for large problems), and the size of the MIP formulation can become prohibitively large in some cases.

Two different MIP solvers can be used: the public-domain glpk solver and the commercial solver. PICO is included in distributions of SPOT.

- **GRASP heuristic** - The 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 ATT commercial solver (**att_grasp**) and an open-source implementation of this solver (**snl_grasp**).

- **Lagrangian Heuristic** - 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.

The following sections provide examples that illustrate the use of the **sp** command. A description of **sp** command line options is available in the **Appendix** (p. 65).

The **sp** command has many different options. The following examples show how different sensor placement optimization problems can be solved with **sp**. Note that these examples can be run in the **C:\spot\examples\simple** directory. The user needs to generate IMPACT files for these examples with the following commands:

```
tevasim --tsg Net3.tsg Net3.inp Net3.out Net3
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.erd
```

5.1 A Simple Example

The following simple example illustrates the way that SPOT has been most commonly used. In this example, SPOT minimizes the extent of contamination (**ec**) while limiting the number of sensors (**ns**) to no more than 5. This problem formulation (eSP) can be efficiently solved with all solvers for modest-size distribution networks, and heuristics can effectively perform sensor placement on very large networks.

We begin by using the PICO solver to solve this problem, with the following command line:

```
sp --path=$bin --path=$pico --path=$mod --network=Net3 --objective=ec --ub=ns,5 --solver=pico
```

This specifies that network **Net3** is analyzed. The objective is to minimize **ec**, the extent of contamination, and an upper-bound of 5 is placed on **ns**, the number of sensors. The solver selected is **pico**, the PICO MIP solver.

This execution of the `sp` command uses the `Net3_ec.impact` file and creates the following files: `Net3.log`, a logfile for the optimization solver, and `Net3.sensors`, a file with the sensor placement locations. Also, `sp` generates the following output:

```
read_impact_files:  Net3_ec.impact

Number of Nodes           :  97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Running glpsol...
... glpsol done
Running PICO...
... PICO done
# No weights file: Net3_ec.impact.prob
```

```
Sensor placement id:      8869
Number of sensors:       5
Total cost:              0
Sensor node IDs:         19 28 54 63 75
Sensor junctions:        119 141 193 207 239

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             8499.9419
Lower quartile impact:   0.0000
Median impact:           6949.0000
Upper quartile impact:   12530.0000
Value at Risk (VaR) (   5%): 25960.0000
TCE                      (   5%): 33323.2833
Max impact:              42994.8000
```

```
Greedy ordering of sensors: Net3_ec.impact
```

```
-1      59035.8890
54      29087.3165
19      18650.2453
63      11492.6496
75      9972.8445
28      8499.9419
Done with sp
```

The initial information up to the statement "... PICO done" is simply output about what solver is run and information from the solver output. The next information beginning with "Sensor placement id:" is generated by `evalsensor` (p.57). This is a summary that describes the sensor placement and the performance of this sensor placement with respect to the impact measure that was minimized. This includes the following data:

- **Sensor placement id** - an integer ID used to distinguish this sensor placement
- **Number of sensors** - the number of sensors in the sensor placement
- **Total cost:** - the cost of the sensor placement, which may be nonzero if cost data is provided
- **Sensor node IDs** - the internal node indexes used by `sp`
- **Sensor junctions** - the EPANET junction labels for the sensor locations

The performance of the sensor placement is summarized for each IMPACT file used with `sp`. The impact statistics included are:

- **min** - The minimum impact over all contamination events. If we make the assumption that a sensor protects the node at which it is placed, then this measure will generally be zero.
- **mean** - The mean (or average) impact over all contamination events.
- **lower quartile** - 25% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **median** - 50% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **upper quartile** - 75% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **VaR** - The value at risk (VaR) uses a user-defined percentile. Given $0.0 < \beta < 1.0$, VaR is the minimum value for which $100 * (1 - \beta)\%$ of contamination events have a smaller impact.
- **TCE** - The tailed-conditioned expectation (TCE) is the mean value of the impacts that are greater than or equal to VaR.
- **worst** - The value of the worst impact.

Finally, a greedy sensor placement is described by `evalsensor`, which takes the five sensor placements and places them one-at-a-time, minimizing the mean impact as each sensor is placed. This gives a sense of the relative priorities for these sensors. The `evalsensor` command can evaluate a sensor placement for a wide variety of different objectives. For example, the command

```
evalsensor --nodemap=Net3.nodemap Net3.sensors Net3_ec.impact \
          Net3_mc.impact Net3_nfd.impact
```

will summarize the solution in the `Net3.sensors` file for the `ec`, `mc` and `nfd` impact measures.

The following example shows how to solve this same problem with the GRASP heuristic. This solver finds the same (optimal) solution as the MIP solver, though much more quickly. The command

```
sp --path=$bin --network=Net3 --objective=ec --ub=ns,5 --solver=snl_grasp
```

generates the following output:

```
read_impact_files: Net3_ec.impact
Note: witness aggregation disabled for grasp

Number of Nodes           : 97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Running iterated descent SNL grasp for *perfect* sensor model
Iterated descent completed
# No weights file: Net3_ec.impact.prob
```

```

Sensor placement id:      8909
Number of sensors:       5
Total cost:              0
Sensor node IDs:         21 33 54 63 75
Sensor junctions:        121 151 193 207 239

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             8656.5521
Lower quartile impact:   0.0000
Median impact:           6480.0000
Upper quartile impact:   13890.0000
Value at Risk (VaR) ( 5%): 28010.0000
TCE ( 5%):              35838.3667
Max impact:              44525.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
54      29087.3165
33      18779.7581
63      11622.1623
75      10102.3572
21      8656.5521
Done with sp

```

Finally, the following example shows how to solve this problem with the Lagrangian heuristic. This solver does not find as good a solution as the GRASP heuristic. The command

```
sp --path=$bin --network=Net3 --objective=ec --ub=ns,5 --solver=lagrangian
```

generates the following output:

```

read_impact_files:  Net3_ec.impact

Number of Nodes      :  97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Setting up Lagrangian data files...
Running UFL solver ...
../.../ bin//ufl Net3_ec.lag  6 0
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      8928
Number of sensors:       5
Total cost:              0
Sensor node IDs:         19 28 54 63 75
Sensor junctions:        119 141 193 207 239

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             8499.9419
Lower quartile impact:   0.0000
Median impact:           6949.0000
Upper quartile impact:   12530.0000

```

```
Value at Risk (VaR) ( 5%): 25960.0000
TCE ( 5%): 33323.2833
Max impact: 42994.8000
```

```
Greedy ordering of sensors: Net3_ec.impact
```

```
-1      59035.8890
54      29087.3165
19      18650.2453
63      11492.6496
75      9972.8445
28      8499.9419
Done with sp
```

5.2 Computing a Bound on the Best Sensor Placement Value

The following example shows how a lower bound can be computed on the best possible sensor placement. That is, any sensor placement would have an expected impact greater than this value. A bound is computed with the following syntax:

```
sp --path=$bin --path=$pico --path=$mod --network=Net3 --objective=ec --ub=ns,5 \
--solver=pico --compute-bound
```

This command generates the following output:

```
read_impact_files: Net3_ec.impact

Number of Nodes      : 97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Running glpsol...
... glpsol done
Running PICO...
... PICO done
Computing a lower bound
Objective lower bound: 8499.94194912
Done with sp
```

5.3 Minimizing the Number of Sensors

The sensor placement problem can be inverted by minimizing the number of sensors subject to a constraint on the extent of contamination. Note that the following example finds a solution with a single sensor that meets our goal of 40000 mean extent of contamination. The command

```
sp --path=$bin --path=$pico --path=$mod --network=Net3 --objective=ns --ub=ec,40000 \
--solver=pico
```

generates the following output:

```
read_impact_files: Net3_ec.impact
```



```

Number of Nodes          : 97
Number of Contamination Impacts: 9480

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Number of sensors=ns
Objective=ns
Statistic=mean
Impact file=Net3_ns.impact
Delay=0

Running glpsol...
... glpsol done
Running PICO...
... PICO done
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      9196
Number of sensors:       1
Total cost:              0
Sensor node IDs:         37
Sensor junctions:        161

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             27097.7191
Lower quartile impact:   3940.0000
Median impact:           22450.0000
Upper quartile impact:   38855.0000
Value at Risk (VaR) ( 5%): 71377.8000
TCE ( 5%):               81046.0667
Max impact:              103746.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

-1      59035.8890
37      27097.7191
Done with sp

```

5.4 Fixing Sensor Placement Locations

Properties of the sensor locations can be specified with the `--sensor-locations` option. This options specifies a **Placement Locations File** (p. 42) that can control whether sensor locations are feasible or infeasible, and fixed or unfixed. For example, suppose the file `locations` contains

```

infeasible 193 119 141 207 239
fixed 161

```

The following example shows how these restrictions impact the solution. Compared to the first example above, we have a less-optimal solution, since the sensor locations above cannot be used and junction 161 must be included. The command

```

sp --path=$bin --path=$pico --path=$mod --network=Net3 --objective=ec --ub=ns,5 \
  --solver=pico --sensor-locations=locations

```

generates the following output:

```

read_impact_files: Net3_ec.impact

```

```

Number of Nodes          : 97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Running glpsol...
... glpsol done
Running PICO...
... PICO done
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      9273
Number of sensors:       5
Total cost:              0
Sensor node IDs:         17 33 37 50 66
Sensor junctions:        115 151 161 185 211

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             9359.6864
Lower quartile impact:   0.0000
Median impact:           7640.0000
Upper quartile impact:   14120.0000
Value at Risk (VaR) ( 5%): 27335.0000
TCE ( 5%):               32282.3000
Max impact:              45300.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
37      27097.7191
66      18228.2936
33      13993.9475
17      11316.9263
50      9359.6864
Done with sp

```

5.5 Robust Optimization of Sensor Locations

The following example demonstrates the optimization of sensor placements using the TCE measure. TCE is the mean value of the worst incidents in the ensemble being evaluated. Given a confidence level $1 - \beta$, TCE is the expectation of the worst impacts with probability β . Compared with our first example, this finds a better solution in terms of TCE, although the mean performance is slightly worse.

The command

```
sp --path=$bin --network=Net3 --objective=ec_tce --ub=ns,5 --solver=snl_grasp
```

generates the following output:

```

read_impact_files: Net3_ec.impact
Note: witness aggregation disabled for grasp

Number of Nodes          : 97
Number of Contamination Impacts: 9480

Number of sensors=5

```

```

Objective=ec
Statistic=tce
Impact file=Net3_ec.impact
Delay=0

Running iterated descent SNL grasp for *perfect* sensor model
Iterated descent completed
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      9289
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:             10287.0856
Lower quartile impact:   1650.0000
Median impact:           10400.0000
Upper quartile impact:   16930.0000
Value at Risk (VaR) (   5%): 24199.0000
TCE                      (   5%): 26376.2167
Max impact:              28564.8000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
88      30705.9487
19      19271.8983
65      12589.0568
17      11151.3907
24      10287.0856
Done with sp

```

Note that the greedy ordering of sensors is less useful in this case. Although we optimized to minimize TCE, the greedy ordering uses the mean value to select each sensor location.

5.6 Multi-Criteria Analysis

The `sp` command supports multi-objective analysis through an iterative process. SPOT does not have a general "pareto search" optimizer. Instead, users can specify constraints with `sp` that ensure that previously optimized objectives are "close" to their previous values. In this way, the user can explicitly search for trade-offs between one-or-more performance objectives.

The examples above consider the extent-of-contamination objective. The sensor placements generated above can be assessed as to how they minimize other objectives like the expected mass of contaminant consumed using `evalsensor`. Consider the solution generated by the previous example (which minimized `ec_tce`), which was copied into the file `Net3_ec.sensors`.

The command

```
evalsensor --nodemap=Net3.nodemap Net3_ec.sensors Net3_mc.impact
```

generates the following output:

```

# No weights file: Net3_mc.impact.prob

```

```

Sensor placement id:      9304

```

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_mc.impact
Number of events:	236
Min impact:	0.0000
Mean impact:	70907.8333
Lower quartile impact:	503.9170
Median impact:	83150.7000
Upper quartile impact:	143999.0000
Value at Risk (VaR) (5%):	144329.0000
TCE (5%):	144555.3333
Max impact:	144732.0000

Greedy ordering of sensors: Net3_mc.impact

-1	137028.5263
65	71611.9658
88	71269.0775
24	71055.0076
17	70965.2738
19	70907.8333

The mean mass consumed is 70907, which is far from the optimal value of 21782 (which we computed separately). The robust optimization example in the previous section is revisited here; "extent of contamination - tce" is still the primary objective, but now a "side constraint" is imposed that precludes any solution that admits an average mass consumed of worse than 30,000 units. The command

```
#!/bin/sh

bin='pwd' /.../.../ bin
mod='pwd' /.../.../ etc/mod
pico='pwd' /.../.../ acro-pico/bin
pythonpath='pwd' /.../.../ python/bin
export PATH=$bin:$pythonpath:$pico:$PATH

if [ ! -e Net3.erd ]; then
# @prelim:
tevasim --tsg Net3.tsg Net3.inp Net3.out Net3
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.erd
# @:prelim
else
cp data/Net3.erd .
cp data/Net3*.erd .
if [ ! -e Net3_mc.impact ]; then
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.erd
fi
fi

# @sp:
sp --path=$bin --network=Net3 --objective=ec_tce --ub=mc_mean,30000 --ub=ns,5 \
--solver=snl_grasp
# @:sp
```

generates the following output:

```
read_impact_files: Net3_ec.impact
read_impact_files: Net3_mc.impact
Note: witness aggregation disabled for grasp

Number of Nodes          : 97
```

```

Number of Contamination Impacts: 9480

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Number of sensors=5
Objective=ec
Statistic=tce
Impact file=Net3_ec.impact
Delay=0

Running iterated descent SNL grasp for *perfect* sensor model
Iterated descent completed
# No weights file: Net3_ec.impact.prob
# No weights file: Net3_mc.impact.prob

```

```

Sensor placement id:      9321
Number of sensors:       5
Total cost:              0
Sensor node IDs:         4 15 29 70 78
Sensor junctions:        35 111 143 219 247

```

```

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             15332.8458
Lower quartile impact:   1650.0000
Median impact:           10660.0000
Upper quartile impact:   26641.8000
Value at Risk (VaR) ( 5%): 40825.8000
TCE ( 5%):               50267.5083
Max impact:              71329.0000

```

```

Impact File:             Net3_mc.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             29350.7370
Lower quartile impact:   133.1430
Median impact:           1277.5100
Upper quartile impact:   23944.9000
Value at Risk (VaR) ( 5%): 144271.0000
TCE ( 5%):               144275.1667
Max impact:              144321.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
4       33458.5542
15      22345.5492
29      17121.4559
78      15503.7144
70      15332.8458

```

```

Greedy ordering of sensors: Net3_mc.impact

```

```

-1      137028.5263
78      58137.1639
29      40727.3168
70      33852.2344
4       29595.1123
15      29350.7370
Done with sp

```

Note that the primary objective, minimizing the TCE of the "extent of contamination" measure, has gotten worse: it is now 50267 rather than 26376. However, our side constraint has been honored, and the mean mass consumed value is now 29350 rather than 70907.

5.7 Sensor Placements without Penalties

A fundamental issue for sensor placement is how to handle the fact that a limited budget of sensors will not be able to cover all possible incidents. SPOT addresses this issue by providing impact measures that integrate an impact 'penalty' for incidents that are not detected by a CWS design. Thus, in the previous examples there was an implicit trade-off between impact reduction and reduction in the number of contamination incidents that are detected.

SPOT also includes impact measures that do not contain these penalties, which allows a user to more directly assess the performance of a CWS design in the context where detections have occurred. For example, the time-to-detection measure (td) includes a penalty for undetected incidents, but the detected-time-to-detection measure (dtd) has no penalty (or, more precisely, a zero penalty).

For example, consider the simple example above, which minimizes the extent of contamination. The **evalsensors** command is applied to the final solution to evaluate the **ec**, **dec** and **nfd** impact measures:

```
evalsensor --nodemap=Net3.nodemap Net3_orig.sensors Net3_ec.impact Net3_dec.impact \
Net3_nfd.impact
```

This generates the following output:

```
# No weights file: Net3_ec.impact.prob
# No weights file: Net3_dec.impact.prob
# No weights file: Net3_nfd.impact.prob

Sensor placement id:      9360
Number of sensors:       5
Total cost:              0
Sensor node IDs:         19 28 54 63 75
Sensor junctions:        119 141 193 207 239

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             8499.9419
Lower quartile impact:   0.0000
Median impact:           6949.0000
Upper quartile impact:   12530.0000
Value at Risk (VaR) ( 5%): 25960.0000
TCE ( 5%): 33323.2833
Max impact:              42994.8000

Impact File:             Net3_dec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             8184.5182
Lower quartile impact:   0.0000
Median impact:           6949.0000
Upper quartile impact:   12530.0000
Value at Risk (VaR) ( 5%): 25960.0000
TCE ( 5%): 33323.2833
Max impact:              42994.8000

Impact File:             Net3_nfd.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             0.2500
Lower quartile impact:   0.0000
Median impact:           0.0000
Upper quartile impact:   0.0000
Value at Risk (VaR) ( 5%): 1.0000
TCE ( 5%): 1.0000
Max impact:              1.0000
```

Greedy ordering of sensors: Net3_ec.impact	
-1	59035.8890
54	29087.3165
19	18650.2453
63	11492.6496
75	9972.8445
28	8499.9419
Greedy ordering of sensors: Net3_dec.impact	
-1	0.0000
19	4845.7771
28	3613.4678
54	10806.3631
63	7097.6114
75	8184.5182
Greedy ordering of sensors: Net3_nfd.impact	
-1	1.0000
75	0.2712
28	0.2500
19	0.2500
54	0.2500
63	0.2500

In this example, the final sensor placement fails to detect 25% of the incidents. It is noteworthy that this does not impact the mean performance very much, since the impact penalty has led to a final solution that fails to detect few incidents with high penalties.

Note that minimizing **dtd** does not really make sense. With zero sensors, you detect no incidents, which means that the final impact measurement is zero! Thus, minimizing **dtd** requires the additional constraint on the number of failed detections (nfd) as well as a limit on the number of sensors (or total sensor costs).

Only the 'pico' SPOT solver currently supports optimization with 'detected' impact measures. For example:

```
sp --path=$bin --path=$pico --path=$mod --network=Net3 --objective=dec --ub=ns,5 \
  --ub=nfd,0.25 --solver=pico
```

generates the following output:

```
read_impact_files: Net3_dec.impact
read_impact_files: Net3_nfd.impact

Number of Nodes          : 97
Number of Contamination Impacts: 9480

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Number of sensors=5
Objective=dec
Statistic=mean
Impact file=Net3_dec.impact
Delay=0

Running glpsol...
... glpsol done
Running PICO...
... PICO done
# No weights file: Net3_dec.impact.prob
```

```
# No weights file: Net3_nfd.impact.prob
```

```
Sensor placement id:      9380
Number of sensors:        5
Total cost:               0
Sensor node IDs:          19 28 54 63 75
Sensor junctions:         119 141 193 207 239
```

```
Impact File:              Net3_dec.impact
Number of events:         236
Min impact:               0.0000
Mean impact:              8184.5182
Lower quartile impact:    0.0000
Median impact:            6949.0000
Upper quartile impact:    12530.0000
Value at Risk (VaR) ( 5%): 25960.0000
TCE ( 5%):                33323.2833
Max impact:               42994.8000
```

```
Impact File:              Net3_nfd.impact
Number of events:         236
Min impact:               0.0000
Mean impact:              0.2500
Lower quartile impact:    0.0000
Median impact:            0.0000
Upper quartile impact:    0.0000
Value at Risk (VaR) ( 5%): 1.0000
TCE ( 5%):                1.0000
Max impact:               1.0000
```

```
Greedy ordering of sensors: Net3_dec.impact
```

```
-1      0.0000
19     4845.7771
28     3613.4678
54     10806.3631
63     7097.6114
75     8184.5182
```

```
Greedy ordering of sensors: Net3_nfd.impact
```

```
-1      1.0000
75      0.2712
28      0.2500
19      0.2500
54      0.2500
63      0.2500
Done with sp
```

5.8 Limited-Memory Sensor Placement Techniques

Controlling memory usage is a critical issue for solving large sensor placement formulations. This is a particular challenge for MIP methods, but both the GRASP and Lagrangian heuristics can have difficulty solving very large problems on standard workstations. A variety of mechanisms have been integrated into **sp** to reduce the problem representation size while preserving the structure of the sensor placement problem. These techniques include: scenario aggregation and filtering, feasible locations, witness aggregation, skeletonization, and memory management.

The **scenarioAggr** (p.62) method described in the previous section is one possible strategy. This tool compresses the impact file while preserving the fundamental structure of the impact file and it is appropriate when optimizing for mean performance objectives. Similarly, the **filter_impacts** (p.59) script can limit the sensor placement to only consider contamination incidents that are "sufficiently bad" in the worst-case.

Another strategy is to limit the number of sensor placements, using the `--sensor-locations` option described above. Eliminating feasible locations reduces the problem representation used by the `sp` solvers.

The *witness aggregation* technique can be used to limit the size of the sensor placement formulation. This term refers to the variables in the MIP formulation that "witness" a contamination event. By default, variables that witness contamination events with the same impact are aggregated, and this typically reduces the MIP constraint matrix by a significant amount. Further reductions can be performed with more aggressive aggregations.

To illustrate the use of witness aggregation, we generated impact files with the `C:\spot\etc\tsg\hourly.tsg` TSG file. The following table illustrates the use of the two witness aggregation options when optimizing the mean extent of contamination: `--aggregation-percent` and `--aggregation-ratio` (used with the `--distinguish-detection` option, which helps this aggregation option). 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.

Aggr Type	Aggr Value	Binary Vars	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

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. The TEVA-SPOT Skeleton code, **spotSkeleton** (p. 73), provides techniques for branch trimming, series pipe merging, and parallel pipe merging. Skeletonization eliminates pipes and junctions that have little impact 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. Details on the two-tiered sensor placement approach can be found in **Section 5.9** (p. 29).

Additionally, the GRASP heuristic has several options for controlling how memory is managed. The `--grasp-representation` 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. It should be noted that the Lagrangian heuristic requires less memory than the GRASP heuristic, and thus similar techniques have not been developed for it.

5.9 Two-tiered Sensor Placement Approach

The two-tiered sensor placement approach uses geographic aggregation on the large impact files to select candidate locations for secondary optimization. While the approach uses detailed hydraulic calculations, it avoids solving sensor placement using the large impact file directly, a process that can exceed the memory available on a personal computer. Geographic aggregation combines witness locations over all scenarios based on their geographic proximity. To maintain hydraulic equivalence as much as possible, skeletonization is used to define geographic proximity. This process is similar to witness aggregation, where witness locations for a single scenario are combined based upon their witness quality. In both cases, the number of possible sensor locations is reduced.

Geographic aggregation reduces the impact file by grouping nodes that are co-located in the skeletonization process. In essence, this reduces the impact file to include only skeletonized nodes (i.e. supernodes) defined in the skeletonized network. From the skeletonized viewpoint, the contaminant hits a supernode when it first crosses the curve defining the supernode. Depending upon the direction the contaminant approaches, it

will hit the real nodes inside the supernodes in different orders. With geographic aggregation, the minimum, maximum, mean, or median could be used to group the time and impact for the real nodes in each supernode. For a sensor on a single real node in the supernode, the maximum statistic is pessimistic for incidents that the node detects. Similarly, using the minimum is always optimistic. For example, consider one supernode that represents four nodes in the original network. The impact values for those nodes are 100, 200, 250, and 300. After geographic aggregation, the impact value is 100 using the minimum, 300 using the maximum, 212.5 using the mean, and 225 using the median. The time associated with each of these impacts is grouped the same way. The new time and impact pair is the aggregated representation of the impact file for that node. For this supernode, four lines of the impact file are reduced to one. This aggregation is performed for each scenario and each supernode. While this greatly reduces the size of the impact file, the number of scenarios remains the same. This method is not influenced by the modified hydraulics of the skeletonized network; rather, the original impact file is modified to reflect geographic aggregation from the skeletonized network.

In the first-tier (Tier 1), sensor placement filters candidate sensor locations based on a geographic aggregation of the impact file. Heuristically, sensor placement in this course search identifies supernodes that are promising regions for sensors. Only real nodes within these selected supernodes are considered candidate sensor locations. All other locations from the impact file generated from the original, most-refined network model are removed. The second-tier (Tier 2) sensor placement uses this refined impact file to place sensors in the original network. Since sensor placement in each tier runs on a smaller impact file than a direct placement on the original network, the two-tiered approach reduces the maximum memory requirement.

The two-tiered sensor placement approach can be run using **sp-2tier** (p. 70). The method calls **spotSkeleton** (p. 73) to define geographic proximity from skeletonized networks. The two-tiered method has been shown to maintain solution quality while greatly reducing the memory footprint need for sensor placement. For additional details, see Klise, Phillips, and Janke [10].

The **sp-2tier** executable is run with the following command line:

```
sp-2tier --path=$bin --network=Net3 --objective=mc --skeleton=8 --stat=min\
--ub1=ns,5 --ub2=ns,5 --solver1=snl_grasp --solver2=snl_grasp
```

This command generates aggregated and refined impact files, along with output files from **sp** based on Tier 1 and Tier 2 sensor placement.

5.10 Evaluating a Sensor Placement

The **evalsensor** (p. 57) executable takes sensor placements in a **Sensor Placement File** (p. 39) and evaluates them using data from an **Impact File** (p. 40) (or a list of impact files). This 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, and if no probabilities are given then uniform probabilities are used by **evalsensor**.

The following example illustrates the use of **evalsensor** after running the first sensor placement optimization example. The command

```
evalsensor --nodemap=Net3.nodemap Net3_orig.sensors Net3_ec.impact Net3_mc.impact
```

generates the following output:

```
# No weights file: Net3_ec.impact.prob
# No weights file: Net3_mc.impact.prob

Sensor placement id:          9404
Number of sensors:           5
```

```

Total cost: 0
Sensor node IDs: 19 28 54 63 75
Sensor junctions: 119 141 193 207 239

Impact File: Net3_ec.impact
Number of events: 236
Min impact: 0.0000
Mean impact: 8499.9419
Lower quartile impact: 0.0000
Median impact: 6949.0000
Upper quartile impact: 12530.0000
Value at Risk (VaR) ( 5%): 25960.0000
TCE ( 5%): 33323.2833
Max impact: 42994.8000

Impact File: Net3_mc.impact
Number of events: 236
Min impact: 0.0000
Mean impact: 43649.6779
Lower quartile impact: 220.0020
Median impact: 1903.6700
Upper quartile impact: 105363.0000
Value at Risk (VaR) ( 5%): 144271.0000
TCE ( 5%): 144327.6667
Max impact: 144477.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
54      29087.3165
19      18650.2453
63      11492.6496
75      9972.8445
28      8499.9419

```

```

Greedy ordering of sensors: Net3_mc.impact

```

```

-1      137028.5263
75      59420.2069
28      44491.5908
63      43867.9752
54      43672.7009
19      43649.6779

```

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 defined by a **Sensor Class File** (p. 43) and a **Junction Class File** (p. 43). Consider the sensor class or sc file, **Net3.imperfectsc**, which defines different categories of sensor failures:

```

1 0.25
2 0.50
3 0.75
4 1.0

```

Sensors of class "1" give false negative readings 25% of the time, sensors of class "2" give them 50% of the time, etc.

Once failure classes have been defined, the junctions of the network are assigned to classes. This is done with a junction class or jc file, like **Net3.imperfectjc** (here we show just the beginning of this file):

```

1 1
2 1

```

```

3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1

```

Given the junction classes, we can run **evalsensor** to determine the expected impact of a sensor placement, given that sensors may fail. Again, using the solution from the original example:

```

evalsensor --nodemap=Net3.nodemap --sc-probabilities=Net3.imperfectsc \
--scs=Net3.imperfectjc Net3_orig.sensors Net3_ec.impact

```

generates the following output:

```

AA
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      9425
Number of sensors:       5
Total cost:              0
Sensor node IDs:         19 28 54 63 75
Sensor junctions:        119 141 193 207 239

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             17193.3515
Lower quartile impact:   3940.0000
Median impact:           15307.2500
Upper quartile impact:   26537.2156
Value at Risk (VaR) ( 5%): 47637.7773
TCE                      ( 5%): 54977.0644
Max impact:              75509.9043

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
54      36574.4596
63      25675.9754
28      21230.2463
75      18841.6488
19      17193.3515

```

Note that the mean impact of this "extent of contamination" changes dramatically if sensors are allowed to fail. The original solution, 8499 pipe feet, was misleading if sensors fail according to the probabilities we have assigned. With sensor failures, the expected impact is 17193 pipe feet -- more than twice the "perfect sensor" impact.

5.11 Sensor Placement with Imperfect Sensors

The GRASP heuristics in SPOT can optimize sensor placements that take into account sensor failures. For example, we can perform sensor placement optimization with imperfect sensors using the `Net3.imperfectsc` and `Net3.imperfectjc` files defined in the previous section. The command

```

sp --path=$bin --network=Net3 --objective=ec --ub=ns,5 --imperfect-scfile=Net3.imperfectsc \
--imperfect-jcfile=Net3.imperfectjc --solver=snl_grasp

```

generates the following output:

```

read_impact_files: Net3_ec.impact
Note: witness aggregation disabled for grasp

Number of Nodes          : 97
Number of Contamination Impacts: 9480

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=Net3_ec.impact
Delay=0

Running iterated descent grasp for *imperfect* sensor model
Iterated descent completed
# No weights file: Net3_ec.impact.prob

```

```

Sensor placement id:      9460
Number of sensors:       5
Total cost:              0
Sensor node IDs:         53 63 75 83 87
Sensor junctions:        191 207 239 257 265

Impact File:             Net3_ec.impact
Number of events:        236
Min impact:              0.0000
Mean impact:             13469.6464
Lower quartile impact:   3610.0000
Median impact:           9640.0000
Upper quartile impact:   20634.3937
Value at Risk (VaR) ( 5%): 34425.0000
TCE ( 5%):               44607.7281
Max impact:              53214.0000

```

```

Greedy ordering of sensors: Net3_ec.impact

```

```

-1      59035.8890
87      28180.3127
63      22144.7653
83      16917.7504
75      15020.8202
53      13469.6464
Done with sp

```

After this optimization, the mean impact is 13469 pipe feet rather than the 17193 pipe feet value for the solution optimized with perfect sensors. Thus, it is clear that the GRASP heuristic makes different choices if the sensors are imperfect.

5.12 Summary of Solver Features

The following table highlights the capabilities of the SPOT optimizers. The previous examples illustrate SPOT's capabilities, but the advanced features in SPOT are not available for all optimizers.

Solver Feature	MIP	GRASP	Lagrangian
Minimize mean impact ¹	YES	YES	YES
Minimize worst impact ²	YES	YES	NO
Minimize number of sensors ³	YES	NO	NO
Robust objectives ⁴	YES	YES	NO
Side-constraints ⁵	YES	YES	YES
Fixed/Invalid locations ⁶	YES	YES	NO
Witness aggregation ⁷	YES	NO	YES
Incident probabilities ⁸	YES	NO	YES
Incident aggregation ⁹	YES	NO	YES
Sparse data management ¹⁰	NO	YES	NO
Imperfect sensor model ¹¹	NO	YES	NO
One imperfect witness ¹²	YES	YES	NO
Computes lower bound ¹³	YES	NO	YES

Footnotes:

1. The mean sensor placement formulation has been tested on a wide range of network models. The GRASP heuristic reduces memory requirements for large problems with reliable solutions. The Lagrangian solver can be used to find a lower bound. The Lagrangian solver solutions generally have larger errors than GRASP. MIP is provably optimal but can be slower and require more memory.
2. Worst case is slow with GRASP. GRASP has much larger error ratios than for mean. The standard MIP formulation is difficult and may not solve. Improvements pending.
3. MIP can be slow for a mean side-constraint. Improvements pending for worst-case side constraints. Other side constraints not tested.
4. VAR and CVAR are slow with GRASP. GRASP has much larger error ratios than for mean. The standard MIP formulation is difficult and may not solve.
5. Not extensively tested. MIP and GRASP side constraints are hard. Lagrangian is soft/heuristic. Side constraint may not be satisfied. Likely worse for main objective as well. Developers recognize the need to tune this.
6. Fixed and invalid locations currently work with both MIP and GRASP solvers. Pending for Lagrangian.
7. Witness aggregation significantly reduces solution quality for Lagrangian, though does reduce memory. No-error aggregation is on by default for MIP.
8. Incident weightings are only implemented for mean or CVAR.
9. Incident aggregation introduces no errors and can reduce space. In initial testing, the prefix property necessary for this aggregation is rare, so this is off by default given the cost of searching and implementing.
10. This is not supported in the SNL GRASP implementation.
11. This is a nonlinear formulation.
12. This is a linear approximation for imperfect sensors implemented with the mean solvers. This is much faster than the GRASP solver for the nonlinear formulation and seems to give values close to the GRASP solution value.
13. Lower bounds provably show how close a solution is to optimal. MIP solvers give gaps or give provably optimal solutions. A lower bound from the Lagrangian solver can validate the quality of a solution (upper bound) from GRASP.

6 File Formats

6.1 TSG File

- **Description:** specifies how an ensemble of EPANET simulations will be performed.
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** tevasim
- **Details:** Each line of a TSG file specifies injection location(s), specie (optional), injection mass, and the injection time-frame: NZD MASS <specie> <injection-mass> <start-time> <end-time>

It is suggested that these files use the *.tsg extension. If <specie> is included, tevasim 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 threat scenario in the ensemble. Each line in the TSG file uses a simple language that is expanded to define the ensemble, as described below. The entire ensemble is comprised of the cumulative effect of all lines in the TSG 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 label identifying one node where contaminant is introduced, 2) The string "ALL" (without quotes), denoting all junction nodes (excluding tanks and reservoirs), 3) The string "NZD", denoting all junction 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 manual for information about these types of water quality sources). [Character string]

<SrcSpecie>: The character ID of the water quality specie added by the source. This parameter must be omitted when using executables built from the standard epanet distribution [Character string]

<SrcStrngth>: The strength of the contaminant source (see Epanet 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 are given below.

One scenario with a single injection at node ID "10", mass rate of 5 mg/min of specie 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 junctions:

```
NZD    MASS    SPECIE1    5    0    1000
```

Multiple scenarios with two injections, one at node ID "10", and the other at all non-zero demand junctions:

```
10    NZD    MASS    SPECIE1    5    0    1000
```

Multiple scenarios with three injections, at all combinations of all junction nodes (if there are N junction nodes, this would generate N3 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>.

6.2 TSI File

- **Description:** specifies how an ensemble of EPANET simulations will be performed.
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** tevasim
- **Details:** It is suggested that these files use the *.tsi extension. The TSI file is generated as output 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 input file specifies the unique attributes of one threat scenario, in the following format. There is no restriction on the number of scenarios.

```
<NodeID1> <SrcTypeIDX1> <SrcSpecieIDX1> <SrcStrngth1> <Start1> <Stop1> <NodeIDN>...
<SrcTypeIDXN> <SrcSpecieIDXN> <SrcStrngthN> <StartN> <StopN>
```

```
<NodeIDi>:      Epanet label identifying the ith node where 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 toolkit documentation fo reference). [Int]
<SrcSpecieIDXi>: The Epanet specie index of the ith contaminant source [Int]
<SrcStrngthi>:  The strength of the ith contaminant source (see Epanet
                  documentation for description of sources). This value
                  represents the product of contaminant flow rate and
                  concentration. [Float]
<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 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 input file, the "baseline source strength" and "source type" will be ignored, but the source pattern will be used, if one is specified.

6.3 DVF File

- **Description:** provides representation of flushing variables (open hydrants and closed pipes)
- **Format:** ascii
- **Created By:** generic_tevasim_flush_ecobj.m (flushing utility)
- **Used By:** tevasim and tso2Impact
- **Details:**

```
DEBUG
```


DETECTION_TIME	RESPONSE_DELAY	VALVE_DELAY	FLUSH_RATE	FLUSH_LENGTH	SIM_LENGTH
NUM_NODES_NN	FLUSH_N1	FLUSH_N2	FLUSH_NN		
NUM_PIPES_NP	PIPE_N1	PIPE_N2	PIPE_NP		
DEBUG	Boolean				
DETECTION_TIME	integer (hours)				
RESPONSE_DELAY	integer (hours)				
VALVE_DELAY	integer (hours)				
FLUSH_RATE	double (native rate)				
FLUSH_LENGTH	integer (hours)				
SIM_DURATION	integer (hours)				
NUM_NODES_NN	integer				
FLUSH_Nx	integer (node index)				
NUM_PIPES_NP	integer				
PIPE_Nx	integer (pipe index)				

6.4 TAI File

- **Description:** describes the information needed for assessing health impacts
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** tevasim
- **Details:** A TAI provides information needed for assessing health impacts. This file is only required for impact values like **pe** that involve health impacts. The following file can be copied directly into a text editor.

```
; THREAT ASSESSMENT INPUT (TAI) FILE
; USAGE: teva-assess.exe <TAI filename>
; Data items explained below
; UPPERCASE items are non-modifiable keywords
; lowercase items are user-supplied values
; | indicates a selection
; ALL is all junction nodes
;
; INPUT-OUTPUT
; * TSONAME      - dirOrFilename:
;                  Name of threat simulator output (ERD or TSO) file if single file
;                  Name of directory if multi-part file
;                  - patternIfDir - pattern to match tso file parts if
;                  using multi-part ERD or TSO file
;                  - version - required only for versions 1 and 2
;                  newer versions contain this information in the header
;                  1 if it is generated from the currentle installed TEVA
;                  distributed version (as of 3/9/2005)
;                  2 if it is generated from Jim's latest code
;                  - storage - required only for versions 1 and 2
;                  0 Original version - all concentration data stored - one
;                  record for each time step
;                  1 Jim's storage reduction scheme - essentially one record
;                  for each node that has at least one non-zero value.
;                  2 Each row of data is further reduced in size by doing
;                  a modified run length encoding.
; * TAONAME      - Name of threat assessment output (TAO) file
; * SAPNAME      - Sensor Analysis Parameter output file (optional)
; * SAPVARS      - Variables to output to SAP file.  Currently supported values:
;                  MeanInfections
;                  MaxInfections
;                  This is required if there is a SAP file specified
;                  The SAP file and vars are only needed when running the
;                  sensor analysis scripts
```

```

;
TSOName      charstring
TAONAME      charstring
SAPNAME      charstring
SAPVARS      varname_1 ... varname_n
;
; 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 prop. 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 and
;                   ATUS MEAN and FIXED5 MEAN
;
DOSETYPE      TOTAL
INGESTIONTYPE DEMAND
INGESTIONRATE value
;
; POPULATION MODEL
; * POPULATION FILE - value is the filename 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 - value is the 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			
DOSE_BINDATA (DOSE RESPONSE) value1 ... value_n			

6.5 ERD File

- **Description:** provides a compact representation of simulation results.
- **Format:** binary
- **Created By:** `tevasim`
- **Used By:** `tso2Impact`
- **Details:** The simulation data generator produces four output file containing the results of all threat simulation scenarios. The database files include an index file (`index.erd`), hydraulics file (`hyd.erd`) and 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:** ERD file format replaced the TSO and SDX file format, created by previous versions of `tevasim`, to extend `tevasim` capability for multi-specie simulation using EPANET-MSX. While `tevasim` produces only ERD files (even for single specie simulation), `tso2Impact` accepts both ERD and TSO file formats.

6.6 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 may have differing numbers of sensors, so each line contains this information. The node indices map to values in the **Node File** (p. 41).

6.7 Impact File

- **Description:** describes the impact of a contamination event at the point that it is witnessed through a water distribution network.
- **Format:** ascii
- **Created By:** tso2Impact
- **Used By:** sp and evalsensor
- **Details:** An IMPACT file describes the impact of a contamination event at the point that it is witnessed throughout a water distribution network. Specifically, the witness events are assumed to occur at junctions in the network.

The first line of an IMPACT file contains the number of events. The next line specifies the types of delayed impacts provided in this file, with the format:

```
<number-of-delays> <delay-time1> ... <delay-timeN>
```

The delay times are in minutes. (Currently, the SPOT utilities only support a single delay time.)

Subsequent lines have the format

```
<scenario-index> <node-index> <time-of-detection> <impact-value>
```

The node index is the index of a witness location for the attack. A scenario ID maps to a line in the network **Scenario File** (p. 41). A node index maps to a line in the network **Node File** (p. 41). The time of detection is in minutes. The value of impacts are in the corresponding units for each impact measure. The different impact measures in each line correspond to the different delays that have been computed.

6.8 LAG File

- **Description:** A sparse matrix file used by the UFL Lagrangian solver
- **Format:** ascii
- **Created By:** createIPData
- **Used By:** lagrangian
- **Details:** This is a variant of the IMPACT format. Conceptually, it can be viewed as a transpose of the matrix specified in an IMPACT file. The first line specifies the number of locations, the number of events, and the impact values:

```
<num-locations> <num-events> <impact>
```

These impact values differ from the values in the IMPACT file, in that they are normalized by the probability of the event. Subsequent lines describe the impact of each event:

```
<location> <event> <impact>
```

Note that the location and event indices are indexed starting at 1.

6.9 Scenario File

- **Description:** The scenario file provides auxiliary information about each contamination incident.
- **Format:** ascii
- **Created By:** tso2Impact
- **Used By:** evalsensor
- **Details:** Each line of this file has the format:

```
<node-index> <EPANET-ID> <source-type> <source-start-time> <source-stop-time> <source-strength>
```

The node index maps to the network **Node File** (p. 41), and the EPANET ID provides this information (redundantly). The scenario start and stop are in minutes, and these values are relative to the start of the EPANET simulation. The source type is the injection mode for an attack, e.g., flow-paced or fixed-concentration. The source strength is the concentration of contaminant at the attack source.

6.10 Node File

- **Description:** provides a mapping from the indices used for sensor placement to the junction IDs used within EPANET
- **Format:** ascii
- **Created By:** tso2Impact
- **Used By:** evalsensor and sensor placement solvers
- **Details:** The node file provides a mapping from the indices used for sensor placement to the IDs used within EPANET. Each line of this file has the format:

```
<node-index> <EPANET-ID>
```

This mapping is generated by **tso2Impact** (p. 76), and all sensor placement solvers subsequently use the node indices internally.

6.11 Sensor Placement Configuration File

- **Description:** a configuration file used to define a sensor placement problem
- **Format:** ascii
- **Created By:** sp
- **Used By:** createIPData
- **Details:** The sensor placement configuration file is generated by the **sp** (p. 65) solver interface, and it contains all of the information that is needed to define a sensor placement problem. The file has the following format:

```
<number-of-junctions> <response-delay>
<number-of-goals>
<goal-name> <goal-filename> <compression-threshold> <compression-percentage> <attack-collapse-flag>
<number-of-measures>\
```

```

<measure-name> ... <measure-name> <objective-flag>\
<bound-value> ... <bound-value>
<fixed-sensor-placements>
<invalid-sensor-placements>
<cost-values>

```

The values in this file correspond to the command-line arguments of the **sp** (p. 65) solver. Compression threshold or percentage refers to node aggregation values. The attack-collapse-flag is a 0 or 1 value in the configuration file, indicating whether compression/aggregation can make an attack trivial (single supernode equivalent to no detection). The <fixed-sensor-placements>, <invalid-sensor-placements> and <cost-values> data sets are simply an import of the data from the corresponding files that are specified within the **sp** (p. 65) solver interface.

6.12 Sensor Placement Costs File

- **Description:** specifies the costs for installing sensors at different junctions throughout a network
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** sp
- **Details:** Each line of this file has the format:

```
<EPANET-ID> <cost>
```

Junctions not explicitly enumerated in this file are assumed to have zero cost unless the ID '___-default___' is specified. For example:

```
___default 1.0
```

This example would specify that all un-enumerated junctions have a cost of 1.0.

6.13 Placement Locations File

- **Description:** specifies whether sensor placements are fixed and whether locations are feasible sensor placement
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** sp
- **Details:** Each line of this file has the format:

```
<keyword> <EPANET-ID> ... <EPANET-ID>
```

The valid keywords are **feasible**, **infeasible**, **fixed** and **unfixed**. These keywords correspond to two semantic states for each location: (1) the feasibility of sensor placement and (2) whether a sensor placement is forced. The semantics of these keywords are as follows:

- **feasible** - the specified locations are **feasible** and **unfixed**
- **infeasible** - the specified locations are **infeasible** and **unfixed**

- **fixed** - the specified locations are constrained to contain a sensor (**fixed** and **feasible**)
- **unfixed** - the specified locations are not constrained to contain a sensor (**unfixed** and **feasible**)

The locations are EPANET-IDs from the network model. Additionally, the keyword **ALL** or ***** can be used to specify that all network locations are to be processed.

A location file is processed sequentially. The initial state is that all locations are feasible and unfixed. Subsequently, each line updates the state of the locations, given the state defined by the previous lines of this file. For example, the file:

```
infeasible ALL
feasible A B C
```

makes all locations infeasible except for locations A, B and C. Similarly

```
fixed ALL
feasible A B C
```

makes all locations fixed except for locations A, B and C; the **feasible** keyword has the same semantics as the **unfixed** keyword.

6.14 Sensor Class File

- **Description:** contains false-negative probabilities for different types of sensors
- **Format:** ascii
- **Created By:** SPOT 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 rate of 25 percent, and a failure class 2 with a false negative rate of 50 percent:

```
1 0.25
2 0.5
....
```

6.15 Junctions Class File

- **Description:** provides the mapping from EPANET junction IDs to failure classes
- **Format:** ascii
- **Created By:** EPANET user
- **Used By:** sp

- **Details:** When a sensor class file is being used, the "junction class" file provides the mapping from junction (node) id's to failure classes. The format of this file is:

```
<node id> <failure class>
<node id> <failure class>
....
```

For example, supposing that junction 1 is of class 2, junction 2 is of class 1, and junction 3 is of class 1:

```
1 2
2 1
3 1
....
```


Bibliography

- [1] R. Bahadur, W. B. Samuels, W. Grayman, D. Amstutz, and J. Pickus. Pipelinenet: A model for monitoring introduced contaminants in a distribution system. In *Proc. World Water and Environmental Resources Congress*. ASCE, 2003.
- [2] J. Berry, R. D. Carr, W. E. Hart, V. J. Leung, C. A. Phillips, and J.-P. Watson. Designing contamination warning systems for municipal water networks using imperfect sensors. *Journal of Water Resources Planning and Management*, 135(4):253–263, 2009.
- [3] J. Berry, L. Fleischer, W. E. Hart, C. A. Phillips, and J.-P. Watson. Sensor placement in municipal water networks. *J. Water Resources Planning and Management*, 131(3):237–243, 2005.
- [4] J. Berry, W. E. Hart, C. A. Phillips, J. Uber, and T. Walski. Water quality sensor placement in water networks with budget constraints. In *Proc. World Water and Environment Resources Conference*, 2005.
- [5] J. Berry, W. E. Hart, C. E. Phillips, J. G. . Uber, and J.-P. Watson. Sensor placement in municipal water networks with temporal integer programming models. *J. Water Resources Planning and Management*, 132(4):218–224, 2006.
- [6] J. W. Berry, W. E. Hart, and C. A. Phillips. Scalability of integer programming computations for sensor placement in municipal water networks. In *Proc. World Water and Environmental Resources Congress*. ASCE, 2005.
- [7] R. Carr, H. Greenberg, W. Hart, G. Konjevod, E. Lauer, H. Lin, T. Morrison, and C. Phillips. Robust optimization of contaminant sensor placement for community water systems. *Mathematical Programming*, 107(1):337–356, 2006.
- [8] J. R. Chastain, Jr. *A Heuristic Methodology for Locating Monitoring Stations to Detect Contamination Events in Potable Water Distribution Systems*. PhD thesis, University of South Florida, 2004.
- [9] W. E. Hart, J. Berry, R. Murray, C. A. Phillips, L. A. Riesen, and J.-P. Watson. SPOT: A sensor placement optimization toolkit for drinking water contaminant warning system design. Technical Report SAND2007-4393, Sandia National Laboratories, 2007.
- [10] K. Klise, C. Phillips, and R. Janke. Two-tiered sensor placement using skeletonized water distribution network models. in review.
- [11] P. Mirchandani and R. Francis, editors. *Discrete Location Theory*. John Wiley and Sons, 1990.
- [12] K. Morley, R. Janke, R. Murray, and K. Fox. Drinking water contamination warning systems: Water utilities driving water research. *J. AWWA*, pages 40–46, 2007.
- [13] R. Murray, J. Berry, and W. E. Hart. Sensor network design for contamination warning systems: Tools and applications. In *Proc. AWWA Water Security Congress*, 2006.
- [14] R. Murray, J. Uber, and R. Janke. Modeling acute health impacts resulting from ingestion of contaminated drinking water. *J. Water Resources Planning and Management: Special Issue on Drinking Water Distribution Systems Security*, 2006.
- [15] A. Ostfeld and E. Salomons. Optimal layout of early warning detection stations for water distribution systems security. *J. Water Resources Planning and Management*, 130(5):377–385, 2004.
- [16] A. Ostfeld, J. G. Uber, E. Salomons, J. W. Berry, W. E. Hart, C. A. Phillips, J.-P. Watson, G. Dorini, P. Jonkergouw, Z. Kapelan, F. di Pierro, S.-T. Khu, D. Savic, D. Eliades, M. Polycarpou, S. R. Ghimire, B. D. Barkdoll, R. Gueli, J. J. Huang, E. A. McBean, W. James, A. Krause, J. Leskovec, S. Isovitsch, J. Xu, C. Guestrin, J. VanBriesen, M. Small, P. Fischbeck, A. Preis, M. Propato, O. Piller, G. B.

- Trachtman, Z. Y. Wu, and T. Walski. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *J Water Resources Planning and Management*, 134(6):556–568, Nov/Dec 2008.
- [17] R. T. Rockafellar and S. Uryasev. Conditional value-at-risk for general loss distributions. *J. of Banking and Finance*, 26(7):1443–1471, 2002.
 - [18] L. A. Rossman. The EPANET programmer’s toolkit for analysis of water distribution systems. In *Proceedings of the Annual Water Resources Planning and Management Conference*, 1999. Available at <http://www.epa.gov/nrmrl/wswrd/dw/epanet.html>.
 - [19] F. Shang, J. Uber, and L. Rossman. Epanet mutli-species extension user’s manual. Technical Report EPA/600/S-07/021, USEPA, 2011.
 - [20] N. Topaloglou, H. Vladimirov, and S. Zenios. CVaR models with selective hedging for international asset allocation. *J. of Banking and Finance*, (26):1535–1561, 2002.
 - [21] G. B. Trachtman. A “strawman” common sense approach for water quality sensor site selection. In *Proc. 8th Annual Water Distribution System Analysis Symposium*, 2006.
 - [22] USEPA. WaterSentinel System Architecture. Technical report, U.S. Environmental Protection Agency, 2005.
 - [23] J.-P. Watson, H. J. Greenberg, and W. E. Hart. A multiple-objective analysis of sensor placement optimization in water networks. In *Proc. World Water and Environment Resources Conference*, 2004.
 - [24] J.-P. Watson, W. E. Hart, and R. Murray. Formulation and optimization of robust sensor placement problems for contaminant warning systems. In *Proc. Water Distribution System Symposium*, 2006.

A Unix Installation

The TEVA-SPOT Toolkit employs the integer programming solvers provided by the Acro software. This example illustrates how to build both SPOT and Acro. After release 2.1.1, SPOT and Acro need to be built and configured separately.

A.1 Downloading

There are several ways that SPOT and Acro can be downloaded. First, these two packages can be downloaded directly from the subversion repository using the `svn` command, which is commonly available on Linux computers. The SPOT and Acro subversion repositories support anonymous checkouts, so you should not need passwords. The following steps checkout the latest trunk version of these packages:

```
svn checkout -q https://software.sandia.gov/svn/public/acro/acro-pico/trunk acro-pico
svn checkout -q https://software.sandia.gov/svn/teva/spot/spot/trunk spot
```

Alternatively, recent tarballs of the `acro-pico` and `spot` software can be downloaded from the Acro download site and SPOT download site. Once downloaded, these compressed tarballs can be expanded using the `tar` command:

```
tar xvfz filename.tar.gz
```

A.2 Configuring and Building

Acro can be configured using the standard build syntax:

```
cd acro-pico
./setup
autoreconf -i -f
./configure
make
```

This builds libraries in `acro-pico/lib`, along with executables in `acro-pico/bin`. The `setup` command bootstraps the configuration process with files from the `bootstrap` directory.

SPOT can be configured and built in a similar manner:

```
cd spot
./setup
autoreconf -i -f
./configure
make
```

B Data Requirements

The Threat Ensemble Vulnerability Assessment Sensor Placement Optimization Tool, TEVA-SPOT, is a software product and decision-making process developed by EPA's TEVA Research program to assist in determining the best location for sensors in a distribution system. In the past, the TEVA-SPOT software has been applied by EPA staff using models and data provided by utilities. In some cases, significant improvements in the models have been made in order to bring them up to the standards required by the TEVA-SPOT software. In order to streamline the application of TEVA-SPOT, this document was developed to describe the requirements and steps that utilities must follow to participate in the TEVA program and/or use TEVA-SPOT.

Table 1 summarizes the data and information required for a water utility to use TEVA-SPOT; each component is described in more detail in the text. In addition to having an appropriate utility network model, utilities will need to make decisions about the nature of the Contamination Warning System they are designing and the types of security incidents that they would like to detect.

Table 1. Information and data required for utilities to design sensor networks using TEVA-SPOT.

INFORMATION AND DATA NEEDED FOR SENSOR PLACEMENT	DESCRIPTION
Utility Network Model	The model (e.g. EPANET input file) should be up-to-date, capable of simulating operations for a 7-10 day period, and calibrated with field data.
Sensor Characteristics	Type of sensors or sampling program, detection limits, and (if applicable) event detection system
Design Basis Threat	Data describing type of event that the utility would like to be able to detect: specific contaminants, behavior of adversary, and customer behavior
Performance Measures	Utility specific critical performance criteria, such as time to detection, number of illnesses, etc.
Utility Response	Plan for response to a positive sensor reading, including total time required for the utility to limit further public exposure.
Potential Sensor Locations	List of all feasible locations for placing sensors, including associated model node/junction.

UTILITY NETWORK MODEL

The TEVA-SPOT software relies upon an EPANET hydraulic model of the network as the mechanism for calculating the impacts resulting from contamination incidents. Therefore, an acceptable model of the distribution system is needed in order to effectively design the sensor system. The following sub-sections describe the various issues/characteristics of an acceptable hydraulic model for use within TEVA-SPOT.

EPANET Software Requirement TEVA-SPOT uses EPANET, a public domain water distribution system modeling software package. In order to utilize TEVA-SPOT, existing network models that were not developed in EPANET must be converted to and be demonstrated to properly operate within EPANET (Version 2.00.11). Most commercial software packages utilize the basic EPANET calculation engine and contain a conversion tool for creating an EPANET input file from the files native to the commercial package. The user may encounter two potential types of problems when they attempt to make the conversion: (1) some commercial packages support component representations that are not directly compatible with EPANET such as representation of variable speed pumps. The representation of these components may need to be

modified in order to operate properly under EPANET; (2) additionally, the conversion programs developed by the commercial software vendors may also introduce some unintended representations within EPANET that may require manual correction. Following conversion, the output from the original model should be compared with the EPANET model output to ensure that the model results are the same or acceptably close (see section on Model Testing).

Extended Period Simulation TEVA-SPOT uses both the hydraulic and water quality modeling portions of EPANET. In order to support the water quality modeling, the model must be an extended period simulation (EPS) that represents the system operation over a period of several days. Typically a model that uses rules to control operations (e.g., turn pump A on when the water level in tank B drops to a specified level) are more resilient and amenable to long duration runs than are those that use controls based solely on time clocks. Model output should be examined to ensure that tank water levels are not systematically increasing or decreasing over the course of the run since that will lead to unsustainable situations.

The required length of simulation depends on the size and operation of the specific water system. However, in general, the length of the simulation should reflect the longest travel times from a source to customer nodes. This can be calculated by running the EPANET model for water age and determining the higher water age areas. In determining the required simulation length, small dead-ends (especially those with zero demand nodes) can be ignored. Typically a run length of 7 to 10 days is required for TEVA-SPOT though shorter periods may suffice for smaller systems and longer run times required for larger or more complex systems.

Seasonal Models In most cases, water security incidents can take place at any time of the day or any season of the year. As a result, sensor systems should be designed to operate during one or more representative periods in the water system. It should be noted that this differs significantly from the normal design criteria for a water system where pipes are sized to accommodate water usage during peak seasons or during unusual events such as fires. In many cases, the only available models are representative of these extreme cases and generally, modifications to such models should be made to reflect a broader time period prior to use with TEVA-SPOT. Specific guidance on selecting models is provided below:

- **Optimal situation:** the utility has multiple models representing average conditions throughout the year, typical higher demand case (e.g., average summer day) and typical lower demand case (e.g., average winter day).
- **Minimal situation:** the utility has a single model representing relatively average conditions throughout the year.
- **Situations to avoid:** the utility has a single model representing an extreme case (e.g., maximum day model).
- **Exceptions:** (1) If a sensor system is being designed to primarily monitor a water system during a specific event such as a major annual festival, then one of the models should reflect conditions during that event; and (2) If the water system experiences little variation in water demand and water system operation over the course of the year, then a single representative model would suffice.

Model Detail A sufficient amount of detail should be represented in the model for use within TEVA-SPOT. This does not mean that an all-pipes model is required nor does it mean that a model that only represents transmission lines would suffice. At a minimum, all parts of the water system that are considered critical from a security standpoint should be included in the model, even if they are on the periphery of the system. The following guidance drawn from the Initial Distribution System Evaluation (IDSE) Guidance Manual of the Final Stage 2 Disinfectants and Disinfection Byproducts Rule provides a reasonable lower limit for the level of detail required for a TEVA-SPOT analysis (USEPA, 2006).

Most distribution system models do not include every pipe in a distribution system. Typically, small pipes near the periphery of the system and other pipes that affect relatively few customers are excluded to a greater or lesser extent depending on the intended use of the model. This process is called skeletonization.

Models including only transmission networks (e.g. pipes larger than 12 inches in diameter only) are highly skeletonized while models including many smaller diameter distribution mains (e.g. 4 to 6 inches in diameter) are less skeletonized. In general, water moves quickly through larger transmission piping and slower through the smaller distribution mains. Therefore, the simulation of water age or water quality requires that the smaller mains be included in the model to fully capture the residence time and potential water quality degradation between the treatment plant and the customer. Minimum requirements for physical system modeling data for the IDSE process are listed below.

- At least 50 percent of total pipe length in the distribution system.
- At least 75 percent of the pipe volume in the distribution system.
- All 12-inch diameter and larger pipes.
- All 8-inch diameter and larger pipes that connect pressure zones, mixing zones from different sources, storage facilities, major demand areas, pumps, and control valves, or are known or expected to be significant conveyors of water.
- All 6-inch diameter and larger pipes that connect remote areas of a distribution system to the main portion of the system or are known or expected to be significant conveyors of water.
- All storage facilities, with controls or settings applied to govern the open/closed status of the facility that reflect standard operations.
- All active pump stations, with realistic controls or settings applied to govern their on/off status that reflect standard operations.
- All active control valves or other system features that could significantly affect the flow of water through the distribution system (e.g., interconnections with other systems, pressure reducing valves between pressure zones).

Model Demands The movement of water through a distribution system is largely driven by water demands (consumption) throughout the system. During higher demand periods, flows and velocities generally increase and vice versa. Demands are usually represented within a model as average or typical demands at most nodes and then (a) global or regional demand multipliers are applied to all nodes to represent periods of higher or lower demand, and (b) temporal demand patterns are applied to define how the demands vary over the course of a day. In some models, demands within a large area have been aggregated and assigned to a central node. In building a model for use with TEVA-SPOT, rather than aggregating the demands and assigning them to only a few nodes, each demand should be assigned to the node that is nearest to the actual point of use. Both EPANET and most commercial software products allow the user to assign multiple demands to a node with different demands assigned to different diurnal patterns. For example, part of the demand at a node could represent residential demand and utilize a pattern representative of residential demand. Another portion of the demand at the same node may represent commercial usage and be assigned to a representative commercial diurnal water use pattern. TEVA-SPOT supports either a single demand or multiple demands assigned to nodes.

Model Calibration/Validation Calibration is the process of adjusting model parameters so that predicted model outputs generally reflect the actual behavior of the system. Validation is the next step after calibration, in which the calibrated model is compared to independent data sets (i.e., data that was not used in the calibration phase) in order to ensure that the model is valid over wider range of conditions. There are no formal standards in the water industry concerning how closely the model results need to match field results nor is there formal guidance on the amount of field data that must be collected. Calibration methods that are frequently used include roughness (c-factor) tests, hydrant flow tests, tracer tests and matching model results over extended periods for pressure, flow and tank water levels to field data collected from SCADA systems or special purpose data collection efforts.

The IDSE Guidance Manual stipulates the following minimum criteria in order to demonstrate calibration. “The model must be calibrated in extended period simulation for at least a 24-hour period. Because storage facilities have such a significant impact upon water age and reliability of water age predictions throughout the distribution system, you must compare and evaluate the model predictions versus the actual water levels of all storage facilities in the system to meet calibration requirements.” For TEVA-SPOT application, the water utility should calibrate the model to a level that they are confident that the model adequately reflects the actual behavior of the water system being represented by the model. Some general guidelines for calibration/validation are shown below:

- If the model has been in operation actively and for several years and has been applied successfully in a variety of extended period simulation situations, then further substantial calibration may not be necessary. However, even in this case, it is prudent to demonstrate the validity of the model by comparing the model results to field measurements such as time-varying tank water levels and/or field pressure measurements.
- If the model has been used primarily for steady state applications, then further calibration/validation emphasizing extended period simulation is needed.
- If the model has been recently developed and not undergone significant application, then a formal calibration/validation process is needed.

Model Tanks Tank mixing models: Most water distribution system models use a “complete mix” tank representation that assumes that tanks are completely and instantaneously mixed. EPANET (and most commercial modeling software models) allow for alternative mixing models such as last in-first out (LIFO), first in-first out (FIFO), and compartment models. If a utility has not previously performed water quality modeling, they may not have determined the most appropriate tank mixing model for each tank. Since the tank mixing model can affect contaminant modeling and thus the sensor placement decisions, tank mixing models should be specified in the EPANET model input files.

Model Testing The final step in preparing the model for use in TEVA-SPOT is to put the model through a series of candidate tests. Following is a list of potential tests that should be considered.

- If the model was developed and applied using a software package other than EPANET, then following its conversion to EPANET, the original model and the new EPANET model should be run in parallel under EPS and the results compared. The models should give virtually the same results or very similar results. Comparisons should include tank water levels and flows in major pipes, pumps and valves over the entire time period of the simulation. If there are significant differences between the two models, then the EPANET model should be modified to better reflect the original model or differences should be explained and justified.
- The EPANET model should be run over an extended period (typically 1 to 2 weeks) to test for sustainability. In a sustainable model, tank water levels cycle over a reasonable range and do not display any systematic drops or increases. Thus, the range of calculated minimum and maximum water levels in all tanks should be approximately the same in the last few days of the simulation as they were in the first few days. Typically, a sustainable model will display results that are in dynamic equilibrium in which temporal tank water level and flow patterns will repeat themselves on a daily (or other periodic) basis.
- If the water system has multiple sources, then the source tracing feature in EPANET should be used to test the movement of water from each source. In most multiple source systems, operators generally have a good idea as to how far the water from each source travels. The model results should be shown to the knowledgeable operators to ensure that the model is operating in a manner that is compatible with their understanding of the system.
- The model should be run for a lengthy period (1 to 2 weeks) using the water age option in EPANET in order to determine travel times. Since the water age in tanks is not usually known before modeling,

a best guess (not zero hours) should be used to set an initial water age for each tank. Then after the long run of the model, a graph of calculated water age should be examined for each tank to ensure that it has reached a dynamic equilibrium and is still not increasing or decreasing. If the water age is still systematically increasing or decreasing, then the plot of age for each tank should be visually extrapolated to estimate an approximate final age and that value should be reinserted in the model as an initial age, and the model rerun for the extended period. The model output of water age should then be investigated for reasonableness, e.g., are there areas where water age seems unreasonably high? This exercise will also help to define a reasonable upper limit for the duration of the model run to be used in the TEVA-SPOT application.

Following these test runs, any identified modifications should be made in the model to ensure that the model will operate properly under TEVA-SPOT. Many utilities will not be able to make all of the above modifications to their network model. In that case, TEVA-SPOT can still be applied; however the overall accuracy of the results will be questionable and should only be considered applicable to the system as described by the model.

SENSOR CHARACTERISTICS

The TEVA-SPOT analysis requires some assumptions about the detection characteristics of the sensors. In particular, the sensor type, detection limit, and accuracy need to be specified. For example, the analysis can specify a contaminant-specific detection limit that reflects the ability of the water quality sensors to detect the contaminant. Alternatively, the analysis can assume perfect sensors that are capable of detecting all non-zero concentrations of contaminants with 100

In order to quantify detection limits for water quality sensors, the utility must indicate the type of water quality sensor being used, as well as the disinfection method used in the system. Generally, water quality sensors are more sensitive to contaminant introduction with chlorine disinfection than with chloramine. As a result, contaminant detection limits may need to be increased in the design of a sensor network for a chloraminated system.

Ongoing pilot studies for EPA's Water Security Initiative use a platform of water quality sensors, including free chlorine residual, total organic carbon (TOC), pH, conductivity, oxidation reduction potential (ORP), and turbidity (USEPA, 2005b). The correlation between contaminant concentration and the change in these water quality parameters can be estimated from experimental data, such as pipe loop studies (Hall et al., 2007; USEPA, 2005c). Of these parameters, chlorine residual and TOC seem to be most likely to detect a wide range of contaminants.

Detection limits for water quality sensors can be defined in terms of the concentration that would change one or more water quality parameters enough that the change would be detected by an event detection system (for example, Cook et al., 2005; McKenna, Wilson and Klise, 2006) or a water utility operator. A utility operator may be able to recognize a possible contamination incident if a change in water quality is significant and rapid. For example, if the chlorine residual decreased by 1 mg/L, the conductivity increased by 150 μ S/cm, or TOC increased by 1 mg/L.

DESIGN BASIS THREAT

A design basis threat identifies the type of threat that a water utility seeks to protect against when designing a CWS. In general, a CWS is designed to protect against contamination threats; however, there are a large number of potentially harmful contaminants and a myriad of ways in which a contaminant can be introduced into a distribution system. Some utilities may wish to design a system that can detect not only high impact incidents, but also low impact incidents that may be caused by accidental backflow or cross-connections. It is critical for a water utility to agree upon the most appropriate design basis threat before completing the sensor network design.

Contamination incidents are specified by the type of contaminant(s), the quantity of contaminant, the location(s) at which the contaminant is introduced into the water distribution system, the time of day of

the introduction, and the duration of the contaminant introduction. Given that it is difficult to predict the behavior of adversaries, it is unlikely that a water utility will know, with any reasonable level of certainty, the specific contamination threats that they may face. The TEVA-SPOT approach assumes that most of these parameter values cannot be known precisely prior to an incident; therefore, the modeling process must take this uncertainty into account.

As an example, probabilities are assigned to each location in a distribution system indicating the likelihood that the contaminant would be introduced at that location. The default assumption is that each location is equally likely to be selected by an adversary (each has the same probability assigned to it). A large number of contamination incidents (an ensemble of incidents) are then simulated and sensor network designs are selected based on how well they perform for the entire set of incidents. Based on their completed vulnerability assessment and other security related studies, a utility may have some knowledge or preconceived ideas that will assist in refining these assumptions. Some specific questions to consider are listed below:

- Are there certain locations that should be assigned higher probabilities? Should all nodes be considered as possible introduction sites or only nodes with consumer demand?
- Should utility infrastructure sites, such as storage tanks, be considered as potential contamination entry locations?
- Are there specific contaminants of concern to this utility?

PERFORMANCE MEASURES

The TEVA-SPOT software utilizes an optimization model that selects the best sensor design in order to meet a specific set of performance measures. These measures are also sometimes referred to as objectives or metrics. There are several performance measures that are currently supported by TEVA-SPOT including:

- the number of persons who become ill from exposure to a contaminant
- the percentage of incidents detected
- the time of detection
- the length of pipe contaminated

Although it requires more time and input from the user, TEVA-SPOT can also consider multiple objectives in its analysis. If the water utility has any preferences in the area of performance measures, they should specify which of the above measures should be used and the relative importance (weight) to be assigned to each measure. If there are other quantitative measures that they wish to be considered, these should be specified.

UTILITY RESPONSE

The TEVA-SPOT analysis uses the concept of “response times” in the analysis of the effectiveness of a sensor system. Response time is an aggregate measure of the total time between the indication of a contamination incident (e.g., detection of an unusual event by a sensor system) and full implementation of an effective response action such that there are no more consequences. The following response activities are likely following sensor detection (USEPA, 2004; Bristow and Brumbelow, 2006):

- Credibility determination: integrating data to improve confidence in detection; for example, by looking for additional sensor detections or detection by a different monitoring strategy, and checking sensor maintenance records.
- Verification of contaminant presence: collection of water samples in the field, field tests and/or laboratory analysis to screen for potential contaminants.

- Public warning: communication of public health notices to prevent further exposure to contaminated water.
- Utility mitigation: implementing appropriate utility actions to reduce likelihood of further exposure, such as isolation of contaminated water in the distribution system or other hydraulic control options.
- Medical mitigation: public health actions to reduce the impacts of exposure, such as providing medical treatment and/or vaccination.

Past analyses have shown that the benefits of a contaminant warning system (CWS) are very dependent on a rapid response time. Typically, the TEVA-SPOT analysis assesses a range of response times between 0 and 24 hours. A zero hour response time is obviously infeasible but is usually analyzed in TEVA-SPOT as the best-case scenario and thus the maximum benefits that can be associated with a CWS. Water utilities should assess their own emergency response procedures and their acceptable risk tolerance in terms of false negative and false positive responses in order to define a range of response times to be used in the TEVA-SPOT analysis.

POTENTIAL SENSOR LOCATIONS

TEVA-SPOT searches potential sensor locations to determine those set of locations (nodes) that will result in the optimal performance measure for a particular number of sensors. Utilities can choose to consider all nodes as potential sensor locations or to limit the search to a subset of specified nodes.

The primary physical requirements for locating sensors at a particular location are accessibility, electricity, physical security, data transmission capability, sewage drains, and temperatures within the manufacturer specified range for the instrumentation (ASCE, 2004). Accessibility is required for installation and maintenance of the sensor stations. Electricity is necessary to power sensors, automated sampling devices, and computerized equipment. Physical security protects the sensors from natural elements and vandalism or theft. Data transmission is needed to transmit sensor signals back to a centralized SCADA database, and can be accomplished through a variety of solutions including digital cellular, private radio, land-line, or fiber-optic cable. Sewage drains are required to dispose of water and reagents from some sensors. Temperature controls may be needed to avoid freezing or heat damage.

Most drinking water utilities can identify many locations satisfying the above requirements, such as pumping stations, tanks, valve stations, or other utility-owned infrastructure. Many additional locations may meet the above requirements for sensor locations or could be easily and inexpensively adapted. Other utility services, such as sewage systems, own sites that likely meet most of the requirements for sensor locations (e.g., collection stations, wastewater treatment facilities, etc.). In addition, many publicly-owned sites could be easily adapted, such as fire and police stations, schools, city and/or county buildings, etc. Finally, many consumer service connections would also meet many of the requirements for sensor placement, although there may be difficulties in securing access to private homes or businesses. Nevertheless, the benefit of using these locations may be worth the added cost. Compliance monitoring locations may also be feasible sites.

The longer the list of potential sensor sites, the more likely one is to design a high-performing CWS. Typically, the TEVA-SPOT analysis will consider two or three subsets of nodes. For example, the set of all nodes in the model, the set of all publicly-owned facilities, and the set of all water utility owned facilities. The utility should develop a list of the (EPANET) node numbers associated with the facilities that they would like to have considered as potential sensor locations. Multiple lists indicating different subsets (such as water utility owned, publicly owned, etc.) may also be specified.

POPULATION

In TEVA-SPOT analyses, the most commonly used performance measure for sensor placement is the number of persons made ill following exposure to contaminated drinking water. In order to estimate health impacts, information is needed on the population at each node. There are a variety of methods that can be used to estimate nodal population.

- The simplest method is to assume a constant relationship between demand assigned to a node and the population for that node. This method is most appropriate for a homogeneous, largely residential area. If a water demand-based population method is to be used, the total population calculated by the model using a per capita water usage rate needs to be verified with the population served considering billing records.
- Alternatively, if the number of service connections and types of service connections (i.e. residential, commercial, industrial, etc.) are known for each node, then this information can be used to estimate population.
- A third alternative involves independent determination of population based on census data. If the use of a census-based population is desired, the population associated with each non-zero demand node of the model needs to be determined using the census data and GIS software. The resulting total population needs to be verified with the population served by the water system.

If the second or third method is used to estimate nodal population, a file (text file or spreadsheet) should be developed listing model node numbers and the population assigned to that node.

For more information about applying the TEVA-SPOT methodology, see Murray et al. (2007) or visit the EPA website <http://www.epa.gov/nhsrsc/water/teva.html>.

ADDITIONAL READING

American Society of Civil Engineers (ASCE), 2004. Interim Voluntary Guidelines for Designing an Online Contaminant Monitoring System.

Berry, J. et al, 2005. Sensor Placement in Municipal Water Networks. *Jour. Wat. Res. Plan. Manag.* 131 (3): 237-243.

Berry, J. et al, 2006. Sensor Placement in Municipal Networks with Temporal Integer Programming Models. *Jour. Wat. Res. Plan. Manag.* 132(4): 218-224.

Bristow, E. and Brumbelow, K., 2006. Delay between sensing and response in water contamination events. *Jour. Infrastructure Systems*, p. 87-95.

Cook, J. et al, 2005. Decision Support System for Water Distribution System Monitoring for Homeland Security. Proceedings of the AWWA Water Security Congress, Oklahoma City.

Hall, J. et al, 2007. On-line Water Quality Parameters as Indicators of Distribution System Contamination. *Jour. AWWA*, 99:1:66-77.

McKenna, S. et al, 2006. Detecting Changes in Water Quality. *Jour. AWWA*, to appear.

Murray, R., Janke, R. Hart, W., Berry, J., Taxon, T. and Uber, J, 2007. A sensor network design decision framework for Contamination Warning Systems. Submitted to *Jour. AWWA*.

U. S. Environmental Protection Agency, 2004. Response Protocol Toolbox: Planning for and Responding to Drinking Water Contamination Threats and Incidents. http://www.epa.gov/safewater/watersecurity/pubs/rptb_response_guidelines.pdf

U. S. Environmental Protection Agency, 2005a. Review of State-of-the-Art Early Warning Systems for Drinking Water.

U. S. Environmental Protection Agency, 2005b. WaterSentinel System Architecture. http://epa.gov/watersecurity/pubs/watersentinel_system_architecture.pdf

U. S. Environmental Protection Agency, 2005c. WaterSentinel Online Water Quality Monitoring as an Indicator of Contamination. http://epa.gov/watersecurity/pubs/watersentinel_wq_monitoring.pdf

U. S. Environmental Protection Agency, 2006. Initial Distribution System Evaluation Guidance Manual for the Final Stage 2 Disinfectants and Disinfection Byproducts Rule. EPA 815-B-06-002. http://www.epa.gov/safewater/disinfection/stage2/pdfs/guide_idse_full.pdf

C Executable evalsensor

C.1 Overview

The **evalsensor** executable is used to compute information about the impact of contamination events for one (or more) sensor placements.

C.2 Command-Line Help

```
Usage: evalsensor [options...] <sensor-file> <impact-file1>
                               [<impact-file2> >...]
Usage: evalsensor [options...] none <impact-file1> [<impact-file2> >...]
A command to read in one or more sensor placements and summarize their
performance according to various metrics.
```

options:

<code>--all-locs-feasible</code>	A boolean flag that indicates that all locations are treated as feasible.
<code>--costs</code>	A file with the cost information for each location id.
<code>--debug</code>	A boolean flag that adds output information about each incident.
<code>--format</code>	The type of output that the evaluation will generate: <code>cout</code> Generates output that is easily read. (default) <code>xls</code> Generates output that is easily imported into a MS Excel spreadsheet. <code>xml</code> Generates an XML-formatted output that is used to communicate with the TEVA GUI. (Not currently supported.)
<code>--gamma</code>	The fraction of the tail distribution used to compute the VaR and TCE performance measures. (The default value is 0.05).
<code>-h, --help</code>	Display usage information
<code>--incident-weights</code>	A file with the weights of the different contamination incidents.
<code>--nodemap</code>	A file with the node map information, for translating node IDs into junction labels.
<code>-r, --responseTime</code>	This parameter indicates the number of minutes that are needed to respond to the detection of a contaminant. As the response time increases, the impact increases because the contaminant affects the network for a greater length of time. Unit: minutes.
<code>--sc-probabilities</code>	A file with the probability of detection for each sensor category.
<code>--scs</code>	A file with the sensor category information for each location id.
<code>--version</code>	Display version information

arguments:

`sensor-file`: A sensor placement file, which contains one or more sensor placements that will be evaluated. If 'none' is specified, then evalsensor will evaluate impacts with no sensors.

`impact-file`: A file that contains the impact data concerning contamination incidents. If one or more impact files are specified, then evaluations are performed for each impact separately.

Note that options like 'responseTime' can be specified with the syntax '--responseTime 10.0' or '--responseTime=10.0'.

C.3 Description

The **evalsensor** executable takes sensor placements in a **Sensor Placement File** (p. 39) and evaluates them using data from an **Impact File** (p. 40) (or a list of impact files). This executable measures the performance of each sensor placement with respect to the set of possible contamination locations.

See **Section 5.9** (p. 30) for further description of this command.

D Executable `filter_impacts`

D.1 Overview

The `filter_impacts` script filters out the low-impact incidents from an impact file.

D.2 Usage

```
filter_impacts [options...] <impact-file> <out-file>
```

D.3 Options

```
--threshold=<val>
Keep only the incidents whose undetected impact is above a
specified threshold.

--percent=<num>
Keep the <num> percent of the incidents with the worst
undetected impact.

--num=<num>
Keep the <num> incidents with the worst
undetected impact.

--rescale
Rescale the impacts using a log10 scale.

--round
Round input values to the nearest integer.
```

D.4 Arguments

```
<impact-file>
The input impact file.

<out-file>
The output impact file.
```

D.5 Description

The `filter_impacts` command reads an impact file, filters out the low-impact incidents, rescales the impact values, and writes out another impact file.

D.6 Notes

None.

E Executable PICO

E.1 Overview

The **PICO** executable used by **sp** that solves linear programs and mixed-integer linear programs.

E.2 Usage

```
PICO [options...] <input-file>
```

E.3 Options

Documentation of **PICO** options is available from the PICO User Manual, which is available from <http://software.sandia.gov/Acro/PICO>.

E.4 Description

PICO is a general-purpose solver for linear and integer programs. This command is not directly used by the user.

PICO uses public-domain software components, and thus it can be used without licensing restrictions. The integer programming format used in SPOT is defined with the AMPL modeling language. PICO integrates the GLPK mathprog problem reader, which is compatible with a subset of the AMPL modeling language. This enables PICO to process an integer programming formulation in SPOT that can also be used with AMPL.

E.5 Notes

- On large-scale tests, we have noted that PICO's performance is often limited by the performance of the public-domain LP solvers that it employs. In some cases, we have noted that these solvers can be over 100 times slower than the state-of-the-art CPLEX LP solver.

F Executable `randomsample`

F.1 Overview

The `randomsample` executable heuristically solves p-median formulations of the sensor placement problem.

F.2 Usage

```
randomsample <sp-configuration-file> <num-sample> <random-seed>  
            <impact-file-representation> <time-limit> [<solution-file>]
```

F.3 Arguments

<sp-configuration-file>

The configuration file generated by the 'sp' script.

<num-sample>

The number of local searches performed by this heuristic.

<random-seed>

A random number seed.

<impact-file-representation>

An integer that indicates how the impact file is stored internally:
0 - sparse and 1 - dense

<time-limit>

A time limit (in seconds) for how long the heuristic should run.

<solution-file>

The name of the output file that contains the solutions found
by this heuristic.

F.4 Description

The `sp` command runs `randomsample` to solve p-median sensor placement problems with the GRASP heuristic. Currently, the following statistics are supported: mean, var (Value At Risk), tce (Tail-Conditional Expectation), and worst-case.

This command is intended to be only used by the `sp` script, which drives both heuristic and exact solvers.

F.5 Notes

- The `randomsample` heuristic currently does not support side constraints other than on the number of sensors. Side-constraints are supported by the `sideconstraints` (p. 64) executable.

G Executable `scenarioAggr`

G.1 Overview

The `scenarioAggr` executable takes an IMPACT file and produces an aggregated impact file.

G.2 Usage

```
scenarioAggr --numEvents=<num_incidents> <impact file>
```

G.3 Options

```
--numEvents=<number>
```

The number of incidents that should be aggregated.

G.4 Description

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** (p. 40)
- `aggrnetwork.impact.prob` - the probabilities of the aggregated incidents. These are non-uniform, so any solver must recognize incident probabilities.

G.5 Notes

- Not all solvers in SPOT can perform optimization with aggregated IMPACT files. In particular, the heuristic GRASP solver does not currently support aggregation because it does not use incident probabilities. The Lagrangian and PICO solvers support incident aggregation. However, initial results suggest that although the number of incidents is reduced significantly, the number of impacts may not be, and solvers may not run much faster.

H Executable createIPData

H.1 Overview

The **createIPData** executable is used by the **sp** solver interface to setup an integer programming formulation for sensor placement.

H.2 Command-Line Help

```
Usage: createIPData [options...] <config-file>
A command to generate AMPL data files , which are used by integer programming
solvers.

options:

    --gamma                The fraction of the tail used to compute VaR and
                           TCE statistics
    -h, --help             Display usage information
    --output               The output file
    --version              Display version information

arguments:

    config-file:  Contains all of the information needed to setup a sensor
                  placement problem.

The createIPData executable is used to setup an integer programming problem for
sensor placement that can be solved using the GeneralSP IP model. The
configuration file input file is generated by the sp optimizationscript. The
output of this executable consists of AMPL data commands that are used to
generate an instance of the GeneralSP IP model.
```

H.3 Description

The input file used by **createIPData** is a **Sensor Placement Configuration File** (p. 41). The output of this executable is sent to standard out, and it is in a format that can be processed by **PICO** (p. 60) and the AMPL solver interface.

H.4 Notes

- This executable is not meant to be run interactively.
- The scalability of this solver has not been well-characterized for large datasets, even when using aggressive aggregation.

I Executable sideconstraints

I.1 Overview

The **sideconstraints** executable heuristically solves p-median formulations of the sensor placement problem where one or more side-constraints are specified. These side constraints are tight, meaning that any solution that violates the side constraints is considered infeasible.

I.2 Usage

```
sideconstraints <sp-configuration-file> <num-samples> <random-seed>  
<impact-file-representation> <time-limit> [<solution-file>]
```

I.3 Arguments

<sp-configuration-file>

The configuration file generated by the ‘sp’ script.

<num-sample>

The number of local searches performed by this heuristic.

<random-seed>

A random number seed.

<impact-file-representation>

An integer that indicates how the impact file is stored internally:

0 - sparse and 1 - dense

<time-limit>

A time limit (in seconds) for how long the heuristic should run.

<solution-file>

The name of the output file that contains the solutions found by this heuristic.

I.4 Description

The **sp** command runs **sideconstraints** to solve p-median sensor placement problems that include side-constraints with the GRASP heuristic. Currently, the following statistics are supported: mean, var (Value At Risk), tce (Tail-Conditional Expectation), and worst-case.

J Executable sp

J.1 Overview

The **sp** executable provides a common interface for sensor placement solvers in TEVA-SPOT.

J.2 Command-Line Help

```
Usage: sp [options]

Options:
  -h, --help                show this help message and exit
  -n NETWORK, --network=NETWORK
                           Name of network file
  --objective=OBJECTIVE
                           Objective names have the form: <goal>_<statistic>
                           ..The objective goals are:
                           ....cost    the cost of sensor placement
                           ....ec      extent of contamination
                           ....dec     detected extent of contamination
                           ....td      time to detection
                           ....dtd     detected time to detection
                           ....mc      mass consumed
                           ....dmc     detected mass consumed
                           ....nfd     number of failed detections
                           ....ns      the number of sensors
                           ....pe      population exposed
                           ....dpe     detected population exposed
                           ....pk      population killed
                           ....dpk     detected population killed
                           ....pd      population dosed
                           ....dpd     detected population dosed
                           ....vc      volume consumed
                           ....dvc     detected volume consumed
                           ..The objective statistics are:
                           ....mean    the mean impact
                           ....median  the median impact
                           ....var     value-at-risk of impact distribution
                           ....tce     tail-conditioned expectation of imp dist
                           ....cvar    approximation to TCE used with IPs
                           ....worst   the worst impact
                           An objective name of the form <goal> is assumed to
                           refer to the objective <goal>_mean. This option may
                           be listed more than once.
  -r DELAY, --responseTime=DELAY
                           This parameter indicates the number of minutes that
                           are needed to respond to the detection of a
                           contaminant. As the response time increases, the
                           impact increases because the contaminant affects the
                           network for a greater length of time. Unit: minutes.
  -g GAMMA, --gamma=GAMMA
                           Specifies the fraction of the distribution of impacts
                           that will be used to compute the var, cvar and tce
                           measures. Gamma is assumed to lie in the interval
                           (0,1]. It can be interpreted as specifying the
                           100*gamma percent of the worst contamination incidents
                           that are used for these calculations. Default: .05
  --incident-weights=INCIDENT_WEIGHTS
                           This parameter specifies a file that contains the
                           weights for contamination incidents. This file
                           supports the optimization of weighted impact metrics.
                           By default, incidents are optimized with weight 1.0
```

`--imperfect-scfile=SCFILE`
 Specifies the name of a file defining detection probabilities for all sensor categories. Used with the `imperfect-sensor` model. Must be specified in conjunction with the `--imperfect-jcfile` option.

`--imperfect-jcfile=JCFILE`
 Specifies the name of a file defining a sensor category for each network junction. Used with the `imperfect-sensor` model. Must be specified in conjunction with the `--imperfect-scfile` option.

`--num=NUMSAMPLES, --numsamples=NUMSAMPLES`
 Specifies the number of candidate solutions generated by the grasp heuristic. Defaults vary based on statistic and sensor model formulation (perfect vs. imperfect).

`--grasp-representation=GRASP_REPRESENTATION`
 Specifies whether the grasp heuristic uses a sparse matrix (0) or dense matrix (1) representation to store the impact file contents. The default is 1.

`--impact-dir=IMPACT_DIRECTORY`
 Specifies the directory the contains impact files. By default the current directory is used.

`--aggregation-threshold=AGGREGATION_THRESHOLD, --threshold=AGGREGATION_THRESHOLD`
 Specifies the value (as '<goal>,<value>') used to aggregate 'similar' impacts. This is used to reduce the total size of the sensor placement formulation (for large problems). The solution generated with non-zero thresholds is not guaranteed to be globally optimal.

`--aggregation-percent=AGGREGATION_PERCENT, --percent=AGGREGATION_PERCENT`
 A '<goal>,<value>' pair where value is a double between 0.0 and 1.0. This is an alternate way to compute the aggregation threshold. Over all contamination incidents, we compute the maximum difference d between the impact of the contamination incident is not detected and the impact it is detected at the earliest possible feasible location. We set the threshold to $d * \text{aggregation_percent}$. If both threshold and percent are set to valid values in the command line, percent takes priority.

`--aggregation-ratio=AGGREGATION_RATIO`
 A '<goal>,<value>' pair where value is a double between 0.0 and 1.0.

`--conserve-memory=MAXIMUM_IMPACTS`
 If location aggregation is chosen, and the original impact files are very large, you can choose to process them in a memory conserving mode. For example "`--conserve_memory=10000`" requests that while original impact files are being processed into smaller aggregated files, no more than 10000 impacts should be read into memory at any one time. Default is 10000 impacts. Set to 0 to turn this off.

`--distinguish-detection=DISTINGUISH_GOAL, --no-event-collapse=DISTINGUISH_GOAL`
 A goal for which aggregation should not allow incidents to become trivial. That is, if the 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. Note: the 'detected' impact measures (e.g. dec, dvc) are always distinguished.

`--disable-aggregation=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.

You may list the goal 'all' to specify all goals.

`--ub-constraint=UB_CONSTRAINT, --ub=UB_CONSTRAINT`
This option specifies a constraint (<objective>,<ub-value>) on the maximal value of an objective type. This option can be repeated multiple times with different objectives.

`--baseline-constraint=BASELINE_CONSTRAINT, --baseline=BASELINE_CONSTRAINT`
Baseline constraints are not currently supported.

`--reduction-constraint=REDUCTION_CONSTRAINT, --reduction=REDUCTION_CONSTRAINT`
Reduction constraints are not currently supported.

`--costs=COST_FILE, --costs_ids=COST_FILE`
This file contains costs for the installation of sensors throughout the distribution network. This file contains id/cost pairs, and default costs can be specified with the id: `__default__`.

`--costs-indices=COST_INDEX_FILE`
This file contains costs for the installation of sensors throughout the distribution network. This file contains index/cost pairs, and default costs can be specified with the index: `-1`.

`--sensor-locations=LOCATIONS_FILE`
This file contains information about whether network ids are feasible for sensor placement, and whether a sensor placement is fixed at a given location.

`--solver=SOLVER`
This option specifies the type of solver that is used to find sensor placement(s). The following solver types are currently supported:
`..att_grasp` multistart local search heuristic (AT&T)
`..snl_grasp` TEVA-SPOT license-free grasp clone
`..lagrangian` lagrangian relaxation heuristic solver
`..pico` mixed-integer programming solver (PICO)
`..glpk` mixed-integer programming solver (GLPK)
`..picoamp` MIP solver with AMPL
`..cplexamp` commercial MIP solver
The default solver is `snl_grasp`.

`--solver-options=SOLVER_OPTIONS`
This option contains solver-specific options for controlling the sensor placement solver. The options are added to the solver command line.

`--runtime=RUNTIME`
Terminate the solver after the specified number of wall clock minutes have elapsed. By default, no limit is placed on the runtime. Some solvers can provide their best solution so far at the point of termination.

`--notify=INTERVAL`
Some solvers can output preliminary solutions while they are running. This option supplies the interval in minutes at which candidate solutions should be printed out.

`--compute-bound`
Only compute a bound on the value of the optimal solution.

`--memmon`
Summarize the maximum memory used by any of the executables

`--memcheck=MEMCHECKTARGET`
This option indicates that valgrind should run on one or more executables.
`..all` run on all executables
`..solver` run on the solver executable
`..createIPData` run on createIPData
`..preprocessImpacts` run on preprocessImpacts
`..evalsensor` run on evalsensor
`..aggregateImpacts` run on aggregateImpacts Output will be written to `memcheck.{name}.{pid}`.

`--tmp-file=TMP_FILE`
Name of temporary file prefix used in this computation. The default name is '`<network-name>`'.

`-o OUTPUT_FILE, --output=OUTPUT_FILE`
Name of the output file that contains the sensor placement. The default name is '`<network-name>.sensors`'.

<code>--summary=SUMMARY_FILE</code>	Name of the output file that contains summary information about the sensor placement.
<code>--format=FORMAT</code>	Format of the summary information
<code>--print-log</code>	Print the solver output
<code>--path=PATH</code>	Add this path to the set of paths searched for executables and IP models.
<code>--amplcplexpath=AMPLCPLEXPAT</code>	Look for <code>ampl</code> and <code>cplexamp</code> executables in this directory. This defaults to a 'blank' path, which implies that the user's system path is used.
<code>--picopath=PICOPATH</code>	Look for the <code>PICO</code> executable in this directory. This defaults to the path used for executables specified by the <code>--path</code> option.
<code>--glpkpath=GLPKPATH</code>	Look for the <code>GLPK</code> executable in this directory. This defaults to the path used for executables specified by the <code>--path</code> option.
<code>--ampl=AMPL</code>	The name of the <code>ampl</code> executable (this defaults to 'ampl').
<code>--ampldata=AMPLDATA</code>	An auxillary <code>AMPL</code> data file. This option is used when integrating auxillary information into the <code>AMPL</code> IP model.
<code>--amplmodel=AMPLMODEL</code>	An alternative <code>AMPL</code> model file. This option is used when applying a non-standard <code>AMPL</code> model for solving sensor placement with an IP.
<code>--seed=SEED</code>	The value of a seed for the random number generator used by the solver. This can be used to ensure a deterministic, repeatable output from the solver. Should be ≥ 1 .
<code>--eval-all</code>	This option specifies that all impact files found will be used to evaluate the final solution(s).
<code>--debug</code>	List status messages while processing.
<code>--gap=GAP</code>	TODO gap help string.
<code>--version</code>	Print version information for the compiled executables used by this command.

J.3 Description

The **sp** executable is a Python script that coordinates the execution of the SPOT sensor placement solvers. The **sp** options can flexibly specify the objective to be optimized, as well as constraints on performance/cost goals.

The **sp** script currently interfaces with integer programming (IP) solvers, GRASP heuristics, and a Lagrangian heuristic. The IP formulation can be used to find globally optimal solutions, the GRASP heuristic has proven effective at finding optimal solutions to a variety of large-scale applications, and the Lagrangian heuristic finds a near-optimal selection while computing a confidence bound.

The following files are generated during the execution of **sp**:

- `<tmpfile>.config` - the **Sensor Placement Configuration File** (p. 41)
- `<tmpfile>.dat` - an `AMPL` data file (if using an IP solver)
- `<tmpfile>.mod` - an `AMPL` script (if using an IP solver)
- `<tmpfile>.log` - a log file that captures the solver output
- `<tmpfile>.sensors` - a **Sensor Placement File** (p. 39) that contains the final solution

J.4 Notes

- The solvers provided by SPOT do not attempt to minimize the number of sensors that are used. This can sometimes lead to confusing behavior, especially for worst-case objectives where there may be different solutions with different numbers of sensors. For small problems, the PICO solver can be used to solve an auxiliary problem, where the number of sensors is minimized subject to the performance value that is found when minimizing impact.
- The heuristic solvers do not currently support the "median" performance measure.
- The IP solvers do not currently support median, var, or tce performance measures.
- The aggregation threshold does not currently impact the problem formulation used by the GRASP heuristic.
- This solver interface assumes that event likelihoods are uniform. The format for specifying non-uniform likelihoods remains unresolved.
- Numerical issues have been observed when solving with the PICO solver in some cases. These usually result in a message that indicates that the solver failed.
- The gamma parameter cannot be varied with `snl_grasp` or `att_grasp`.
- The `snl_grasp` and `att_grasp` solvers cannot effectively minimize the worst-case objective when the problem is constrained.
- The "ub" option to `sp` is a misnomer when the Lagrangian solver is selected. The side constraints are really goal constraints, and therefore the values specified are not true upper bounds. However, we have decided to keep a consistent usage for both side and goal constraints rather than introducing a new option.
- The Lagrangian heuristic can now be used with "witness aggregated" problems as the PICO solver can. This cuts down the required memory in a dramatic way. For example, a problem that caused the Lagrangian to use 1.8 GB of RAM and run in 20 minutes when unaggregated, was solved with only 27MB of RAM in 20 seconds when aggregated. There is a disadvantage, though. The actual quality of the witness-aggregated solution of the Lagrangian solver can be 25 percent (or more) worse than the unaggregated solution. This could be improved in the future.
- The `sp` executable currently defaults to invoke witness aggregation when the Lagrangian solver is selected. If you want to turn this feature off, you must use the `disable-aggregation` option. To disable aggregation of all objectives, use the option `disable-aggregation=all`, as in the example above.

K Executable sp-2tier

K.1 Overview

The **sp-2tier** executable provides a interface for sensor placement using a two-tiered approach in TEVA-SPOT. This executable calls the **sp** script. In tier 1, **sp-2tier** transforms the impact file using geographic aggregation. The aggregated impact file is used to filter out candidate sensor placement locations. In tier 2, **sp-2tier** refines the original impact file to include only candidate locations for the final sensor placement. **sp-2tier** uses **spotSkeleton** to define the geographic proximity used to transform the original impact file. See [Section 5.9](#) (p. 29). for additional information on the two-tiered sensor placement approach.

K.2 Command-Line Help

```
Usage: sp_2tier [options]

Options:
  -h, --help                show this help message and exit
  -n NETWORK, --network=NETWORK
                           Name of network file
  --objective=OBJECTIVE    Objective names have the form: <goal>_<statistic>
                           ..The objective goals are:
                           ....cost    the cost of sensor placement
                           ....ec      extent of contamination
                           ....dec     detected extent of contamination
                           ....td      time to detection
                           ....dtd     detected time to detection
                           ....mc      mass consumed
                           ....dmc     detected mass consumed
                           ....nfd     number of failed detections
                           ....ns      the number of sensors
                           ....pe      population exposed
                           ....dpe     detected population exposed
                           ....pk      population killed
                           ....dpk     detected population killed
                           ....pd      population dosed
                           ....dpd     detected population dosed
                           ....vc      volume consumed
                           ....dvc     detected volume consumed
                           ..The objective statistics are:
                           ....mean    the mean impact
                           ....median  the median impact
                           ....var     value-at-risk of impact distribution
                           ....tce     tail-conditioned expectation of imp dist
                           ....cvar    approximation to TCE used with IPs
                           ....worst   the worst impact
                           An objective name of the form <goal> is assumed to
                           refer to the objective <goal>_mean.
  --skeleton=SKELETON      Specifies the pipe diameter threshold used in
                           spotSkeleton.
  --stat=STAT              Statistic used in geographic aggregation of the input
                           file. Options include:
                           ...min
                           ...max
                           ...mean
                           ...median
                           Streaming algorithms are used to aggregate the impact
                           file. For the median option, the impact file must be
                           sorted.
  --ub1-constraint=UB1_CONSTR, --ub1=UB1_CONSTR
```

	This option specifies a constraint (<objective>,<ub-value>) on the maximal value of an objective type in tier 1.
<code>--ub2-constraint=UB2_CONSTRAINT</code>	<code>--ub2=UB2_CONSTRAINT</code> This option specifies a constraint (<objective>,<ub-value>) on the maximal value of an objective type in tier 2.
<code>--solver1=SOLVER1</code>	This option specifies the type of solver that is used to find sensor placement(s) in tier 1. The following solver types are currently supported: ..att_grasp multistart local search heuristic (AT&T) ..snl_grasp TEVA-SPOT license-free grasp clone ..lagrangian lagrangian relaxation heuristic solver ..pico mixed-integer programming solver (PICO) ..glpk mixed-integer programming solver (GLPK) ..picoamp MIP solver with AMPL ..cplexamp commercial MIP solver The default solver is <code>snl_grasp</code> .
<code>--solver2=SOLVER2</code>	This option specifies the type of solver that is used to find sensor placment(s) in tier 2. The following solver types are currently supported: ..att_grasp multistart local search heuristic (AT&T) ..snl_grasp TEVA-SPOT license-free grasp clone ..lagrangian lagrangian relaxation heuristic solver ..pico mixed-integer programming solver (PICO) ..glpk mixed-integer programming solver (GLPK) ..picoamp MIP solver with AMPL ..cplexamp commercial MIP solver The default solver is <code>snl_grasp</code> .
<code>--path=PATH</code>	Add this path to the set of paths searched for executables and IP models.

K.3 Description

Based on the command line naming conventions, the following files are required for execution of **sp-2tier**:

INPUT FILE	DESCRIPTION
<code>NETWORK.inp</code>	Original EPANET input file
<code>NETWORK_OBJECTIVE</code> <code>.impact</code>	Impact file for a single objective, output from <code>tevasim/tso2impact</code> using the original EPANET input file.
<code>NETWORK.nodemap</code>	Nodemap file used to translate between <code>epanetID</code> (used in the <code>inp</code> file) and <code>nodeID</code> (used in the <code>impact</code> file).

From the data directory, **sp-2tier** creates a results directory called `NETWORK_SKELETON`. For example, `Net3_8` contains output files from **sp-2tier** using `Net3` and a skeleton threshold of 8 inches. The following files are generated during the execution of **sp-2tier**:

OUTPUT FILE	DESCRIPTION
NETWORK__SKELETON.inp	Skeletonized EPANET inp file
NETWORK__SKELETON.map	Map file (notation in epanetID)
SKELETON__time.out	Time used to run spotSkeleton
SKELETON__memmon.out	Memory used to run spotSkeleton
NETWORK__SKELETON__OBJECTIVE.impact	Impact file after geographic aggregation
NETWORK__SKELETON.nodemap	Nodemap file after geographic aggregation
aggregate__time.out	Time used to aggregate the impact file
aggregate__memmon.out	Memory used to aggregate the impact file
NETWORK__SKELETON.log	sp log file for tier 1 sensor placement
NETWORK__SKELETON.sensors	sp sensors file for tier 1 sensor placement
NETWORK__SKELETON.config	sp config file for tier 1 sensor placement
sp1__SKELETON.out	sp output file for tier 1 sensor placement. Contains memory used.
sp1__time.out	Time used to run tier 1 sensor placement
NETWORK__SKELETONR__OBJECTIVE.impact	Refined impact file based on sp1 result
NETWORK__SKELETONR.nodemap	Refined nodemap file. This file contains 4 columns (as opposed to the standard 2 for nodemap files) and is used to convert between refined nodeID and epanetID on the original network. Col1 = nodeID in the refined impact file. Col2 = supernode membership. Col3 = nodeID in the original impact file. Col4 = epanetID in the original inp file
refine__time.out	Time used to refine the impact file
refine__memmon.out	Memory used to refine the impact file
NETWORK__SKELETONR.log	sp log file for tier 2 sensor placement
NETWORK__SKELETONR.sensors	sp sensors file for tier 2 sensor placement
NETWORK__SKELETONR.config	sp config file for tier 2 sensor placement
sp2__SKELETONR.out	sp output file for tier 2 sensor placement. Contains memory used.
sp2__time.out	Time used to run tier 2 sensor placement

K.4 Notes

- **sp-2tier** will not recreate the skeletonized inp and map files if the file NETWORK__SKELETON.map is in the results directory NETWORK__SKELETON.
- **sp-2tier** will not recreate the aggregated impact file if the file NETWORK__SKELETON__OBJECTIVE.impact is in the results directory NETWORK__SKELETON.

L Executable **spotSkeleton**

L.1 Overview

The TEVA-SPOT skeletonizer, **spotSkeleton**, reduces the size of a network model by grouping neighboring nodes based on the topology of the network and pipe diameter threshold. **spotSkeleton** requires an EPANET inp network file and a pipe diameter threshold along with output file names for the skeletonized EPANET inp network file and map file. **spotSkeleton** contracts connected pieces of the network into a single node; this node is called a supernode. The map file defines the nodes that belong to each supernode.

L.2 Usage

```
spotSkeleton <input inp file> <pipe diameter threshold> <output inp file> <output map file>
```

L.3 Description

spotSkeleton includes branch trimming, series pipe merging, and parallel pipe merging. A pipe diameter threshold determines candidate pipes for skeleton steps. **spotSkeleton** 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. **spotSkeleton** repeats these steps until no further pipes can be removed from the network. **spotSkeleton** creates an EPANET-compatible skeletonized network inp file and a 'map file' that contains the mapping of original network model nodes into skeletonized supernodes.

Under the skeletonization steps described above, 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.

L.4 Notes

- Commercial skeletonization codes include Haestad Skelebrator, MWHSoft H2OMAP, and MWHSoft InfoWater. Two-tiered sensor placement requires mapping between the skeletonized and original network (map file). Commercial skeletonizers do not provide this information directly. The MWHSoft Infowater/H2OMAP history log file provides it indirectly in a history log file. The history log file tracks sequential skeleton steps and records merge and trim operations on a pipe-by-pipe basis.
- To validate **spotSkeleton**, we compared its output to that 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 TEVA-SPOT skeletonizers were compared using the Jaccard index (Jaccard, 1901). 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, 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 TEVA-SPOT

skeletonizers result in a Jaccard index between 0.93 and 0.95. Thus, we believe the **spotSkeleton** is a good substitute for commercial skeletonizers.

M Executable tevasim

M.1 Overview

The **tevasim** executable uses EPANET to perform an ensemble of contaminant transport simulations.

M.2 Command-Line Help

```
Usage: tevasim [options] <epanet-input-file> <epanet-output-file>
        <erd-db-name>
```

A utility for running an ensemble of water quality simulations, whose results are stored in a TSO file.

options:

<code>--dvf</code>	The Decision Variable File (DVF) file used to specify the flushing/valve controls.
<code>-h, --help</code>	Display usage information
<code>--mss</code>	The MSX species to save.
<code>--msx</code>	The MSX file for specifying multi-species EPANET.
<code>--tsg</code>	The TSG file used to specify the injection incidents.
<code>--tsi</code>	The TSI file used to specify the injection incidents.
<code>--version</code>	Display version information

arguments:

`epanet-input-file`: EPANET network file.

`epanet-output-file`: Output file generated by EPANET.

`erd-db-name`: The ERD database name. A directory can be specified as part of this name — if there is one, the db files will be stored there

The **tevasim** command is used to simulate contamination incidents. This command uses EPANET to perform an ensemble of contaminant transport simulations, defined by a TSG File. The following files are generated during the execution of **tevasim**:

- a binary ERD file that contains the contamination transport data,
- a binary SDX file that provides an index into the ERD File, and
- an output file that provides a textual summary of the EPANET simulations.

Note that options like ‘tsg’ can be specified with the syntax ‘`--tsg file.tsg`’ or ‘`--tsg=file.tsg`’.

N Executable tso2Impact

N.1 Overview

The **tso2Impact** executable generates one or more impact files from a ERD or TSO file.

N.2 Command-Line Help

```
Usage: tso2Impact [options] <output-prefix> <tso-directory-or-files>
      <tai-directory-or-files>
An application that reads a TSO file (and associated TAI file if health
impacts are to be computed) and creates one or more impact files that are used
to formulate sensor placement problems.

options:

      --dec                If this option is specified, an impact file will
                          be generated for the 'detected extent of
                          contamination' measure.
      --detectionConfidence The number of sensors that must detect an event
                          before the impacts are calculated. Normally
                          this is 1 sensor.
-d, --detectionLimit      A list of thresholds needed to perform detection
                          with a sensor. There must be one threshold for
                          each .tso file. The units of these detection
                          limits depend on the units of the contaminant
                          simulated for each TSO file (e.g. number of
                          cells of a biological agent).
      --dmc                If this option is specified, an impact file will
                          be generated for the 'detected mass consumed'
                          measure.
      --dpd                If this option is specified, an impact file will
                          be generated for the 'detected population dosed'
                          measure. This is an intensive measure to
                          compute.
      --dpe                If this option is specified, an impact file will
                          be generated for the 'detected population
                          exposed' measure. This is an intensive measure
                          to compute.
      --dpk                If this option is specified, an impact file will
                          be generated for the 'detected population
                          killed' measure. This is an intensive measure
                          to compute.
      --dtd                If this option is specified, an impact file will
                          be generated for the 'detected
                          time-to-detection' measure.
      --dvc                If this option is specified, an impact file will
                          be generated for the 'detected volume consumed'
                          measure.
      --dvf                This is used to correct demands generated for
                          flushing response calculations
      --ec                If this option is specified, an impact file will
                          be generated for the 'extent of contamination'
                          measure.
-h, --help                Display usage information
      --mc                If this option is specified, an impact file will
                          be generated for the 'mass consumed' measure.
      --nfd                If this option is specified, an impact file will
                          be generated for the 'number-of-failed
                          detections' measure.
      --pd                If this option is specified, an impact file will
                          be generated for the 'population dosed' measure.
```


<code>--pe</code>	This is an intensive measure to compute. If this option is specified, an impact file will be generated for the 'population exposed' measure. This is an intensive measure to compute.
<code>--pk</code>	If this option is specified, an impact file will be generated for the 'population killed' measure. This is an intensive measure to compute.
<code>-r, --responseTime</code>	This option indicates the number of minutes that are needed to respond to the detection of a contaminant. As the response time increases, the impact increases because the contaminant affects the network for a greater length of time. Unit: minutes.
<code>--species</code>	Name of species to use in computing impacts.
<code>--td</code>	If this option is specified, an impact file will be generated for the 'time-to-detection' measure.
<code>--tec</code>	If this option is specified, an impact file will be generated for the 'timed extent of contamination' measure.
<code>--vc</code>	If this option is specified, an impact file will be generated for the 'volume consumed' measure.
<code>--version</code>	Display version information

arguments:

`output-prefix`: The prefix used for all files generated by `tso2Impact`.

`erd-or-tso-file`: This argument indicates either a TSO or ERD file.

`tai-directory-or-file`: This argument indicates a TAI file name. The TAI input file is a `threat_assess` input that specifies parameters like dosage, response, lethality, etc. There should be one TAI file for each TSO file.

Note that options like 'responseTime' can be specified with the syntax '`--responseTime 10.0`' or '`--responseTime=10.0`'.

N.3 Description

The **tso2Impact** executable generates impact files that are used for sensor placement. This executable processes a **ERD or TSO File** (p. 39), which summarizes the result of an EPANET computation. The following files are generated during the execution of **tso2Impact**:

- `<output-prefix>.nodemap` - a **Node File** (p. 41).
- `<output-prefix>.scenariomap` - a **Scenario File** (p. 41).
- `<output-prefix>_<impact-type>.impact` - an **Impact File** (p. 40) for a given impact.

N.4 Notes

- The '`--tsoPattern`' option allows a set of ERD or TSO files to be specified without explicitly listing all of them on the command-line. The user specifies a regular expression, and all files that match that expression are included in the analysis.

O Executable `ufl`

O.1 Overview

The `ufl` executable heuristically solves p-median formulations of the sensor placement problem while also computing a valid lower bound on the best possible sensor placement value.

O.2 Usage

```
ufl <sp-configuration-file> <p> [--gap=<fraction>][<goal-constraint-data-file> <upper-bound>]
```

O.3 Options

`--gap=<fraction>`

This option tells the solver to stop when the solution is within a certain percentage of optimal. Let `\b icost` be the current best integer solution found and `\b lb` be the current lower bound. The solver will stop with ` (icost - lb)/lb ` is less than the gap. For example, if the gap is 0.1, then the solver will stop when it has a solution that is within 10 percent of optimality.

O.4 Arguments

`<sp-configuration-file>`

A LAG file that defines impacts for the objective.

`<p>`

The number of sensors.

`<goal-constraint-data-file>`

A LAG file that defines impacts for a side-constraint.

`<upper-bound>`

The upper bound for this side constraint.

O.5 Description

"ufl" stands for "uncapacitated facility location," and this code is a Sandia-modified version of the combination of Lagrangian relaxation and the "Volume Algorithm" that is found in the open-source "COIN" repository (that the PICO solver uses).

The `sp` executable automatically generates `ufl` commands, including those with goal constraints. The user specifies the number of sensors, and `sp` passes to `ufl` one more than this number. The Lagrangian heuristic implemented in `ufl` then places the correct number of sensors, and one "dummy" sensor that catches all undetected events.

Note that the `ufl` command uses **LAG File** (p. 40) inputs, which are a modified format of impact files. These files are generated by the **tso2Impact** (p. 76) executable.

As of `teva-spot-1.2`, `ufl` handles "goal constraints." For example, we may minimize the contaminant mass consumed subject to the goal of limiting the extent of contamination in pipe feet to a constant such as

15,000. This is different from specifying a side constraint for the "sideconstraints" local search executable. The latter will reject any solution in which the extent of contamination is greater than 15,000, even if it is only 15,001. Many goal constraints may be provided simultaneously, and the Lagrangian solver will attempt to find a solution that honors those constraints. It will report one that has a good combination of primary objective value and small violations of the goals.

This technology is young, and experience shows that user attempts to make the goal constraints too tight can confuse the solver. We offer the following guidance to avoid this problem. Suppose that we wish to use the Lagrangian heuristic to find a good solution that minimizes the average contaminant mass consumed subject to utility guidelines on the average extent of contamination, and also the average volume of contaminated water consumed.

1. Using a solver of choice for the particular problem, find single-objective optimal values for each objective.
2. Using `evalsensor`, evaluate the single-objective sensor placements against each of the other objectives. The result is a matrix of objective values.
3. Determine goal constraints for the secondary objectives by selecting a value between the optimal single-objective value for that secondary objective, and its value under the sensor placement obtained by solving the single-objective problem for the primary objective.

For example, for a real test problem, minimizing the average contaminant mass consumed yielded an objective value of 638,344 units. Taking the sensor placement obtained from that solve, we found that the average extent of contamination was 78,037 feet, and the average volume of contaminated water consumed was 282,689 units.

Solving individually for these objectives, we found that the optimal solutions for extent of contamination and volume consumed were 40,867 and 217,001, respectively. From this information, we decided to apply goal constraints of 45,000 feet for the extent of contamination, and 250,000 units for the volume consumed.

Minimizing the mass consumed with these two goal constraints, the Lagrangian heuristic found a new sensor placement that incurred objective values of 678,175 units for mass consumed, 49,016 feet for the extent of contamination, and 256,615 units for volume consumed. Note that neither goal was strictly met, but each goal helped improve its related objective value.

We now compare this technology to the side-constrained local search heuristic (the "sideconstraints" executable). Each heuristic has advantages and disadvantages. The goal-constrained Lagrangian solver can handle an arbitrary number of goal constraints, producing a solution that is well balanced, as above. When we attempt to reproduce the results above using the "sideconstraints" executable, which is currently limited to only one side constraint, we see the untreated objective suffer. For example, with the same setup as above, and a side constraint of 50,000 feet for average extent of contamination, the sideconstraints heuristic produces a solution with an expected mass consumed of 670,399 units, an expected extent of contamination of 49,827 feet, and an expected volume consumed of 326,943. We see a similar type of result with a single side constraint on the volume consumed (the extent of contamination increases substantially). The sideconstraints code could be extended to handle multiple side constraints, of course, but the neighborhood search might have difficulty finding feasible solutions. Since Lagrangian relaxation is a global technique and slightly infeasible solutions are permitted, we are more likely to find a good trade-off.

However, the Lagrangian heuristic has disadvantages as well. If a particular goal constraint is set too tightly, the solution can degenerate such that all of the objectives get substantially worse. We do not understand this phenomenon well yet, and further research into the algorithm itself may be necessary to make this technology generally usable. For now, it is sometimes necessary to manipulate the values of the goal constraints manually in order to find a good solution.

SCIENCE



PRESORTED STANDARD
POSTAGE & FEES PAID
EPA
PERMIT NO. G-35

Office of Research and Development
National Homeland Security Research Center
Cincinnati, OH 45268

Official Business
Penalty for Private Use
\$300



Recycled/Recyclable
Printed with vegetable-based ink on
paper that contains a minimum of
50% post-consumer fiber content
processed chlorine free