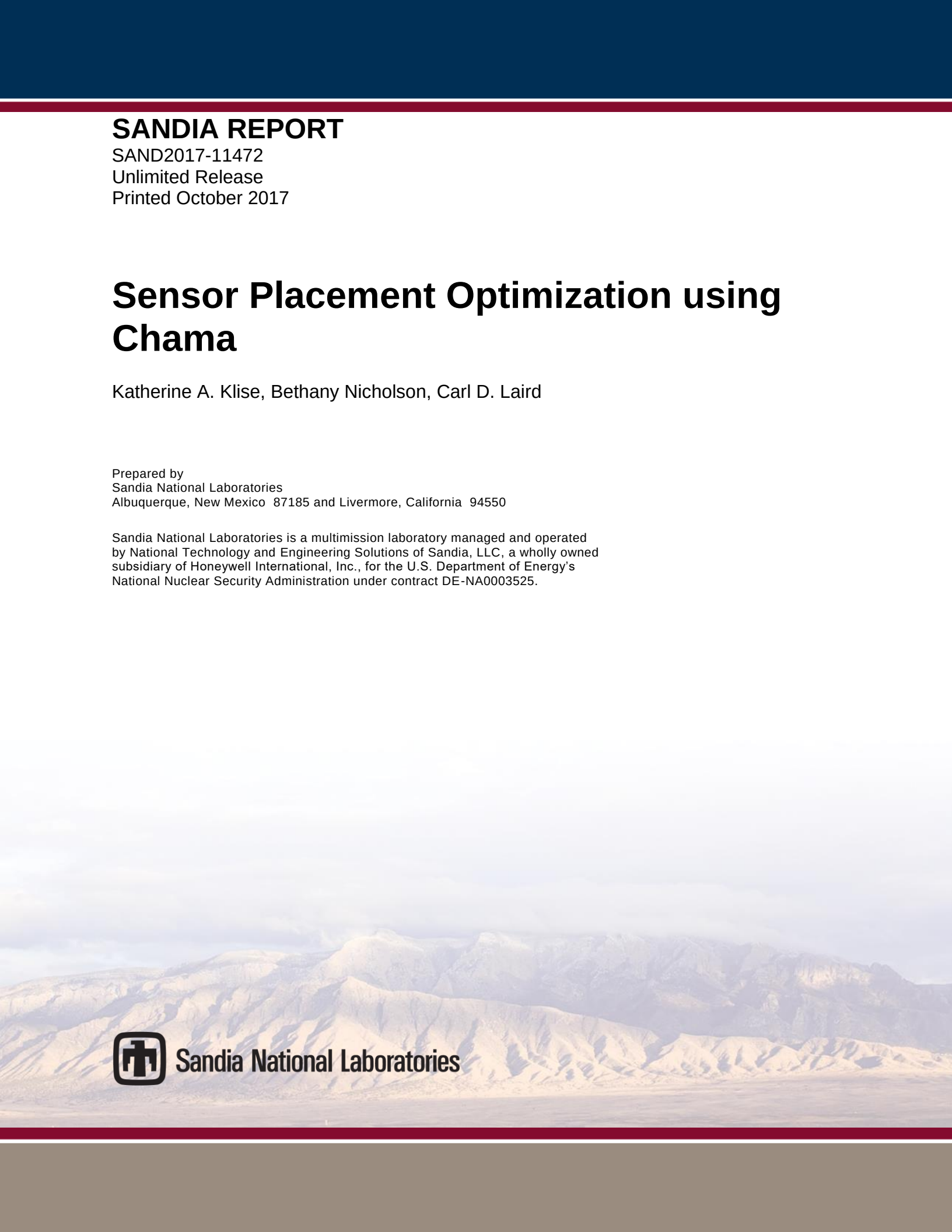
SANDIA REPORT
SAND2017-11472
Unlimited Release
Printed October 2017

Sensor Placement Optimization using Chama

Katherine A. Klise, Bethany Nicholson, Carl D. Laird

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

SAND2017-11472
Printed October 2017
Unlimited Release

Sensor Placement Optimization using Chama

Katherine A. Klise[1], Bethany Nicholson[2], Carl D. Laird[2]
[1] Geotechnology and Engineering Department, [2] Discrete Math and Optimization Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185-MS0751

Abstract

Continuous or regularly scheduled monitoring has the potential to quickly identify changes in the environment. However, even with low-cost sensors, only a limited number of sensors can be deployed. The physical placement of these sensors, along with the sensor technology and operating conditions, can have a large impact on the performance of a monitoring strategy.

Chama is an open source Python package which includes mixed-integer, stochastic programming formulations to determine sensor locations and technology that maximize monitoring effectiveness. The methods in Chama are general and can be applied to a wide range of applications. Chama is currently being used to design sensor networks to monitor airborne pollutants and to monitor water quality in water distribution systems. The following documentation includes installation instructions and examples, description of software features, and software license. The software is intended to be used by regulatory agencies, industry, and the research community. It is assumed that the reader is familiar with the Python Programming Language. References are included for additional background on software components. Online documentation, hosted at http://chama.readthedocs.io/, will be updated as new features are added. The online version includes API documentation.

Contents

List of Figures

1 Overview

Chama is an open source Python package which includes sensor placement optimization methods for a wide range of applications. Some of the methods in Chama were originally developed by Sandia National Laboratories and the U.S. Environmental Protection Agency to design sensor networks to detect contamination in water distribution systems [13] [14]. In this context, contamination scenarios are precomputed using a water network model, feasible sensor locations and thresholds are defined, and the optimization method selects a set of sensors to minimize a given objective.

Chama was developed to be a general purpose sensor placement optimization software tool. The software includes mixed-integer, stochastic programming formulations to determine sensor locations and technology that maximize monitoring effectiveness. The software is intended to be used by regulatory agencies, industry, and the research community. Chama expands on previous software tools by allowing the user to optimize both the location and type of sensors in a monitoring system. Chama includes functionality to define point and camera sensors that can be stationary or mobile. Furthermore, transport simulations can represent a wide range of applications, including (but not limited to):

- Atmospheric dispersion
- Liquid and gas transport through pipe networks
- Surface and ground water transport
- Seismic wave propagation

The basic steps required for sensor placement optimization using Chama are shown in Figure 1.

Figure 1: Basic steps in sensor placement optimization using Chama

- *Transport simulation*: Generate an ensemble of transport simulations representative of the system in which sensors will be deployed.
- *Sensor technology*: Define a set of feasible sensor technologies, including stationary and mobile sensors, point detectors and cameras.
- *Impact assessment*: Extract the impact of detecting transport simulations given a set of sensor technologies.
- *Optimization*: Optimize sensor location and type given a sensor budget.
- *Graphics*: Generate maps of the site that include the optimal sensor layout and information about scenarios that were and were not detected.

The user can enter the workflow at any stage. For example, if the impact assessment was determined using other methods, Chama can still be used to optimize sensor placement. The following documentation includes additional information on these steps, along with installation instructions, software application programming interface (API), and software license. It is assumed that the reader is familiar with the Python Programming Language. References are included for additional background on software components.

2 Installation

Chama requires Python (2.7, 3.4, 3.5, or 3.6) along with several Python package dependencies. Information on installing and using Python can be found at https://www.python.org/. Python distributions, such as Anaconda, are recommended to manage the Python interface.

To install the latest stable version of Chama using pip:

```
pip install chama
```

To install the development branch of Chama from source using git:

```
git clone https://github.com/sandialabs/chama
cd chama
python setup.py install
```

Developers should build Chama using the setup.py 'develop' option.

2.1 Dependencies

Required Python package dependencies include:

- Pyomo [2]: formulate optimization methods, https://github.com/pyomo.
- Pandas [8]: analyze and store databases, http://pandas.pydata.org.
- Numpy [15]: support large, multi-dimensional arrays and matrices, http://www.numpy.org.
- Scipy [15]: support efficient routines for numerical analysis, http://www.scipy.org.

Optional Python package dependencies include:

- Matplotlib [3]: produce graphics, http://matplotlib.org.
- nose: run software tests, http://nose.readthedocs.io.

3 Transport simulation

Chama requires a set of precomputed transport simulations to determine optimal sensor placement. The type of transport simulation depends on the application and scale of interest. Multiple scenarios should be generated to capture uncertainty in the system. For each scenario, a **signal** is recorded.

For example:

- **To place sensors to detect a gas leak**, an atmospheric dispersion model can be used to simulate gas concentrations. Multiple scenarios capture uncertainty in the leak rate, leak location, wind speed and direction. Depending on the region of interest and the complexity of the system, very detailed or simple models can be used. In this case, the **signal** is concentration.
- **To place sensors to detect contaminant in a water distribution system**, a water network model can be used to simulate hydraulics and water quality. Multiple scenarios capture uncertainty in the location, rate, start time, and duration of the injection along with uncertainty in customer demands. EPANET [10], WNTR [4], or similar water network simulators, can be used to run this type of analysis. In this case, the **signal** is concentration.
- **To place sensors to detect a seismic event**, a wave propagation model can be used to simulate displacement. Multiple scenarios capture uncertainty in the location and magnitude of the seismic event along with subsurface heterogeneity. Depending on the region of interest and the complexity of the system, very detailed or simple models can be used. In this case, the **signal** is displacement.

For each scenario, the time, location, and signal are recorded. The points used to record time and location can be sparse to help reduce data size. Chama uses Pandas DataFrames [8] to store the signal data. Pandas includes many functions to easily populate DataFrames from a wide range of file formats. For example, DataFrames can be generated from Excel, CSV, and SQL files. Signal data can be stored in XYZ or Node format, as described below.

3.1 XYZ format

In XYZ format, the X, Y, and Z location is stored for each entry. In the DataFrame, X, Y, and Z describe the location, T is the simulation time, and Sn is the signal for scenario n. Exact column names must be used for X, Y, Z, and T. The scenario names can be defined by the user. When using this format, Chama can interpolate sensor measurements that are not represented in the signal data. An example signal DataFrame in XYZ format is shown below using a simple 2x2x2 system with three time steps and fabricated data for three scenarios.

```
>>> print(signal)
    X  Y  Z   T    S1    S2    S3
0   1  1  1   0  0.00  0.00  0.00
1   1  1  1  10  0.00  0.00  0.01
2   1  1  1  20  0.00  0.00  0.00
3   2  1  1   0  0.25  0.21  0.20
4   2  1  1  10  0.32  0.14  0.25
5   2  1  1  20  0.45  0.58  0.61
6   1  2  1   0  0.23  0.47  0.32
7   1  2  1  10  0.64  0.12  0.15
8   1  2  1  20  0.25  0.54  0.24
9   2  2  1   0  0.44  0.15  0.45
10  2  2  1  10  0.25  0.28  0.68
11  2  2  1  20  0.82  0.12  0.13
12  1  1  2   0  0.96  0.53  0.64
13  1  1  2  10  0.61  0.23  0.21
14  1  1  2  20  0.92  0.82  0.92
15  2  1  2   0  0.41  0.84  0.75
16  2  1  2  10  0.42  0.87  0.98
17  2  1  2  20  0.00  0.51  0.55
18  1  2  2   0  0.00  0.00  0.13
19  1  2  2  10  0.00  0.00  0.00
20  1  2  2  20  0.00  0.00  0.00
21  2  2  2   0  0.00  0.00  0.00
22  2  2  2  10  0.00  0.00  0.00
23  2  2  2  20  0.00  0.00  0.00
```

3.2 Node format

In Node format, a location index is stored for each entry. The index can be a string, integer, or float. This format is useful when working with sparse systems, such as nodes in a networks. In the DataFrame, Node is the location index, T is the simulation time, and Sn is the signal for scenario n. Exact column names must be used for Node and T. The scenario names can be defined by the user. When using this format, Chama does not interpolate sensor measurements and only stationary point sensors can be used to extract detection time. An example signal DataFrame in Node format is shown below using 4 nodes with three time steps and fabricated data for three scenarios.

```
>>> print(signal)
   Node   T    S1    S2    S3
0    n1   0  0.00  0.00  0.00
1    n1  10  0.32  0.14  0.25
2    n1  20  0.25  0.54  0.24
3    n2   0  0.00  0.00  0.01
4    n2  10  0.45  0.58  0.61
5    n2  20  0.44  0.15  0.45
6    n3   0  0.00  0.00  0.00
7    n3  10  0.23  0.47  0.32
8    n3  20  0.25  0.28  0.68
9    n4   0  0.25  0.21  0.20
10   n4  10  0.64  0.12  0.15
11   n4  20  0.82  0.12  0.13
```

3.3 Internal simulation engines

Chama includes methods to run simple Gaussian plume and Gaussian puff atmospheric dispersion models [1]. Both models assume that atmospheric dispersion follows a Gaussian distribution. Gaussian plume models are typically used to model steady state plumes, while Gaussian puff models are used to model non-continuous sources. The `chama.transport` module has additional information on running the Gaussian plume and Gaussian puff models. Note that many atmospheric dispersion applications require more sophisticated models.

The following simple example runs a single Gaussian plume model for a given receptor grid, source, and atmospheric conditions.

Import the required Python packages:

```
>>> import numpy as np
>>> import pandas as pd
>>> import chama
```

Define the receptor grid:

```
>>> x_grid = np.linspace(-100, 100, 21)
>>> y_grid = np.linspace(-100, 100, 21)
>>> z_grid = np.linspace(0, 40, 21)
>>> grid = chama.transport.Grid(x_grid, y_grid, z_grid)
```

Define the source:

```
>>> source = chama.transport.Source(-20, 20, 1, 1.5)
```

Define the atmospheric conditions:

```
>>> atm = pd.DataFrame({'Wind Direction': [45, 60],
...                     'Wind Speed': [1.2, 1],
...                     'Stability Class': ['A', 'A']}, index=[0, 10])
```

Initialize the Gaussian plume model and run (the first 5 rows of the signal DataFrame are printed):

```
>>> gauss_plume = chama.transport.GaussianPlume(grid, source, atm)
>>> gauss_plume.run()
```

```
>>> signal = gauss_plume.conc
>>> print(signal.head(5))
       X      Y    Z  T    S
0 -100.0 -100.0  0.0  0  0.0
1 -100.0 -100.0  2.0  0  0.0
2 -100.0 -100.0  4.0  0  0.0
3 -100.0 -100.0  6.0  0  0.0
4 -100.0 -100.0  8.0  0  0.0
```

The Gaussian Puff model is run in a similar manner. The time between puffs (tpuff) and time at the end of the simulation (tend) must be defined.

Initialize the Gaussian puff model and run:

```
>>> gauss_puff = chama.transport.GaussianPuff(grid, source, atm, tpuff=1, tend=10)
>>> gauss_puff.run(grid, 10)
>>> signal = gauss_puff.conc
```

3.4 External simulation engines

Transport simulations can also be generated from a wide range of external simulation engines, for example, atmospheric dispersion can be simulated using AERMOD [12] or CALPUFF [11] or using detailed CFD models, transport in pipe networks can be simulated using EPANET [10] or WNTR [4], and groundwater transport can be simulated using MODFLOW [7]. Output from external simulation engines can be easily formatted and imported into Chama.

4 Sensor technology

Many different sensor technologies exist. For example, in the context of gas detection, sensors can monitor the concentration at a fixed point or they can be based on optical gas imaging technology and monitor an area within a field of view. Sensors can monitor continuously or at defined sampling times. Sensors can also be mounted on vehicles or drones and move through a specified region. Furthermore, sensors can have different operating conditions which can change detectability. In order to understand the tradeoffs between different sensor technologies and operating conditions and to select an optimal subset of sensors, these different options should be considered simultaneously within an optimal sensor placement problem.

The `chama.sensors` module can be used to define sensor technologies in Chama. The module is used to represent a variety of sensor properties including detector type, detection threshold, location, and sampling times. Additionally, every sensor object includes a function that accepts a *signal*, described in the *Transport simulation* section, and returns the subset of that signal that is detected by a set of sensors. This information is then used to extract the *impact* of each sensor on each scenario, as described in the *Impact assessment* section. The sensor placement optimization uses this measure of 'impact' to select sensors.

Each sensor is declared by specifying a **position** and a **detector**. The following options are available in Chama (additional sensor technologies could easily be incorporated).

4.1 Position options

- **Stationary**: A stationary sensor that is fixed at a single location.
- **Mobile**: A mobile sensor that moves according to defined waypoints and speed. It can also be defined to repeat its path or start moving at a particular time. A mobile sensor is assumed to be at its first waypoint for all times before its starting time and is assumed to be at its final waypoint if it has completed its path and the repeat path option was not set.

4.2 Detector options

- **Point**: A point detector. This type of detector determines detection by comparing a signal to the detector's threshold.
- **Camera**: A camera detector using the camera model from [9]. This type of detector determines detection by collecting the signal within the camera's field of view, converting that signal to pixels, and comparing that to the detector's threshold in terms of pixels.

When using signal data in XYZ format, Chama can interpolate sensor measurements that are not represented in the signal data. However, the sample time of a Camera detectors must be represented in the signal data (i.e. only X, Y, and Z can be interpolated).

For example, a **stationary point sensor**, can be defined as follows:

```
>>> pos1 = chama.sensors.Stationary(location=(1,2,3))
>>> det1 = chama.sensors.Point(threshold=0.001, sample_times=[0,2,4,6,8,10])
>>> stationary_pt_sensor = chama.sensors.Sensor(position=pos1, detector=det1)
```

A **mobile point sensor**, can be defined as follows:

```
>>> pos2 = chama.sensors.Mobile(locations=[(0,0,0),(1,0,0),(1,3,0),(1,2,1)],speed=1.
↪2)
>>> det2 = chama.sensors.Point(threshold=0.001, sample_times=[0,1,2,3,4,5,6,7,8,9,
↪10])
>>> mobile_pt_sensor = chama.sensors.Sensor(position=pos2, detector=det2)
```

A **stationary camera sensor**, can be defined as follows:

```
>>> pos3 = chama.sensors.Stationary(location=(2,2,1))
>>> det3 = chama.sensors.Camera(threshold=400, sample_times=[0,5,10], direction=(1,1,
↪1))
>>> stationary_camera_sensor = chama.sensors.Sensor(position=pos3, detector=det3)
```

A **mobile camera sensor**, can be defined as follows:

```
>>> pos4 = chama.sensors.Mobile(locations=[(0,1,1),(0.1,1.2,1),(1,3,0),(1,2,1)],
↪speed=0.5)
>>> det4 = chama.sensors.Camera(threshold=100, sample_times=[0,3,6,9], direction=(1,
↪1,1))
>>> mobile_camera_sensor = chama.sensors.Sensor(position=pos4, detector=det4)
```

When using signal data in Node format, Chama does not interpolate sensor measurements that are not represented in the signal data and only stationary point sensor can be used. When using Node format, a **stationary point sensor**, can be defined as follows:

```
>>> pos1 = chama.sensors.Stationary(location='Node1')
>>> det1 = chama.sensors.Point(threshold=0.001, sample_times=[0,2,4,6,8,10])
>>> stationary_pt_sensor = chama.sensors.Sensor(position=pos1, detector=det1)
```

Note that the units for time, location, speed, and threshold need to match the units from the transport simulation.

5 Impact assessment

Impact assessment extracts the **impact** if a particular sensor detects a particular scenario. Impact can be measured using a wide range of metrics. In Chama, impact assessment starts by extracting the times when each sensor detects a transport scenario. Detection can be used in a wide range of sensor placement optimizations, including maximizing coverage or minimizing detection time. The `chama.impact` module is used to extract detection times and convert detection time to other damage metrics.

Chama uses Pandas DataFrames [8] to store the impact assessment. Each DataFrame has three columns: Scenario, Sensor, and Impact. Exact column names must be used. Note that the values in the Impact column can represent different metrics.

5.1 Detection times

In general, detection depends on the scenario environmental conditions, the sensor location, and sensor operating conditions. While some scenarios can be detected by a single sensor multiple times, other scenarios can go undetected by all sensors.

The following example demonstrates how to extract detection times using a predefined signal, computed using the *Transport simulation* module, and a set of predefined sensors, constructed using the *Sensor technology* module.

Group sensors in a dictionary:

```
>>> sensors = {}
>>> sensors['A'] = stationary_pt_sensor
>>> sensors['B'] = mobile_pt_sensor
>>> sensors['C'] = stationary_camera_sensor
>>> sensors['D'] = mobile_camera_sensor
```

Extract detection times:

```
>>> det_times = chama.impact.detection_times(signal, sensors)
```

```
>>> print(det_times)
  Scenario Sensor             Impact
0       S1      A               [30]
1       S1      B               [30]
2       S1      C  [10, 20, 30, 40]
3       S2      A      [10, 20, 30]
4       S2      B          [20, 30]
5       S2      C  [10, 20, 30, 40]
6       S3      A          [20, 30]
7       S3      B          [20, 30]
8       S3      C      [20, 30, 40]
```

The example shows that Scenario S1 was detected by Sensor A at time 30 (units of time depend on the transport simulation). Scenario S1 was also detected by Sensor B and time 30 and Sensor C at times 10, 20, 30 and 40. Scenario S2 was detected by Sensors A, B, and C. Scenario S3 was detected by Sensors A, B, and C. Sensor D did not detect any scenarios.

This information can be used directly to optimization a sensor layout that maximizes coverage. To optimize a sensor layout that minimizes detection time, each detected scenario-sensor pair must be represented by a single detection time. This can be obtained by taking the minimum, mean, or median from the list of detection times.

Extract the minimum detection time:

```
>>> min_det_time = chama.impact.detection_time_stats(det_times, 'min')
>>> print(min_det_time)
  Scenario Sensor  Impact
0       S1      A      30
1       S1      B      30
2       S1      C      10
3       S2      A      10
4       S2      B      20
```

```
5        S2       C       10
6        S3       A       20
7        S3       B       20
8        S3       C       20
```

5.2 Damage metrics

Depending on the information available from the transport simulation, detection time can be converted to other measures of damage, such as damage cost, extent of contamination, or ability to protect critical assets and populations. These metrics can be used in sensor placement optimization to minimize damage. For example, if the cost of detecting scenario S1 at time 30 is $80,000, then the damage metric for that scenario can be translated from a detection time of 30 to a cost of $80,000. The data associated with damage is stored in a Pandas DataFrame with one column for time (T) and one column for each scenario (name specified by the user).

Example damage costs, associated with each scenario and time:

```
>>> print(damage_cost)
    T       S1      S2       S3
0   0        0       0        0
1  10    10000    5000    15000
2  20    40000   20000    50000
3  30    80000   75000    95000
4  40   100000   90000   150000
```

Convert detection time to damage cost:

```
>>> damage_metric = chama.impact.translate(min_det_time, damage_cost)
>>> print(damage_metric)
  Scenario Sensor  Impact
0       S1      A   80000
1       S1      B   80000
2       S1      C   10000
3       S2      A    5000
4       S2      B   20000
5       S2      C    5000
6       S3      A   50000
7       S3      B   50000
8       S3      C   50000
```

Note that the ‘translate’ function interpolates based on time, if needed. The damage metric can be used in sensor placement optimization to minimize damage.

6 Optimization

The `chama.optimize` module contains **P-median** and **coverage** sensor placement optimization. Additional methods could be added to this module.

6.1 P-median

The P-median formulation is used to determine optimal sensor placement and type that minimizes impact, where impact can be detection time or some other measure of damage. The P-median formulation is written in Pyomo [2] and solved using open source or commercial solvers. The open source GLPK solver [6] is used by default. The P-median sensor placement formulation is described below:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{a \in A} \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} & \\
\text{subject to} \quad & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 & \forall a \in A \\
& x_{ai} \leq s_i & \forall a \in A, i \in \mathcal{L}_a \\
& \sum_{i \in L} c_i s_i \leq p & \\
& s_i \in \{0, 1\} & \forall i \in L \\
& 0 \leq x_{ai} \leq 1 & \forall a \in A, i \in \mathcal{L}_a
\end{aligned}
$$

where:

- A is the set of all scenarios
- L is the set of all candidate sensors
- $\mathcal{L}_\dashv$ is the set of all sensors that are capable of detecting scenario a
- α_a is the probability of occurrence for scenario a
- d_{ai} is the impact coefficient, and represents some measure of the impact that will be incurred if scenario a is first detected by sensor i
- x_{ai} is an indicator variable that will be 1 if sensor i is installed and that sensor is the first to detect scenario a (where *first* is defined as the minimum possible impact, usually defined as time to detection)
- s_i is a binary variable that will be 1 if sensor i is selected, and 0 otherwise
- c_i is the cost of sensor i
- p is the sensors budget

The size of the optimization problem is determined by the number of binary variables. Although x_{ai} is a binary indicator variable, it is relaxed to be continuous between 0 and 1, and yet it always converges to a value of 0 or 1. Therefore, the number of binary variables that need to be considered by the solver is a function of the number of candidate sensors alone, and not the number of scenarios considered. This formulation has been used to place sensors in large water distribution networks [13] and [14] and for gas detection in petrochemical facilities [5].

The user supplies the impact assessment, d_{ai}, sensor budget, p, and (optionally) sensor cost, c_i and the scenario probability, α_a, as described below:

- Impact assessment: A single detection time (or other measure of damage) for each sensor that detects a scenario. Impact is stored as a Pandas DataFrmae, as described in the *Impact assessment* section.
- Sensor budget: The number of sensors to place, or total budget for sensors. If the 'use_sensor_cost' flag is True, the sensor budget is a dollar amount and the optimization uses the cost of individual sensors. If the 'use_sensor_cost' flag is False (default), the sensor budget is a number of sensors and the optimization does not use sensor cost.
- Sensor characteristics: Sensor characteristics include the cost of each sensor. Sensor characteristics are stored as a Pandas DataFrame with columns 'Sensor' and 'Cost'. Cost is used in the sensor placement optimization if the 'use_sensor_cost' flag is set to True.

- Scenario characteristics: Scenario characteristics include scenario probability and the impact for undetected scenarios. Scenario characteristics are stored as a Pandas DataFrame with columns 'Scenario', 'Undetected Impact', and 'Probability'. Undetected Impact is required for each scenario. When minimizing detection time, the undetected impact value can be set to a value larger than time horizon used for the study. Individual scenarios can also be given different undetected impact values. Probability is used if the 'use_scenario_probability' flag is set to True.

Results are stored in a dictionary with the following information:

- Sensors: A list of selected sensors
- Objective: The expected (mean) impact based on the selected sensors
- Assessment: The impact value for each sensor-scenario pair. The assessment is stored as a Pandas DataFrame with columns 'Scenario', 'Sensor', and 'Impact' (same format as the input Impact assessment') If the selected sensors did not detect a particular scenario, the impact is set to the Undetected Impact.

The following example demonstrates the use of P-median sensor placement:

```
>>> print(min_det_time)
  Scenario Sensor  Impact
0       S1      A     2.0
1       S2      A     3.0
2       S3      B     4.0
>>> print(sensor)
  Sensor    Cost
0      A   100.0
1      B   200.0
2      C   500.0
3      D  1500.0
>>> print(scenario)
  Scenario  Undetected Impact  Probability
0       S1                48.0         0.25
1       S2               250.0         0.60
2       S3               100.0         0.15

>>> pmedian = chama.optimize.Pmedian(use_scenario_probability=True, use_sensor_
↪cost=True)
>>> results = pmedian.solve(min_det_time, 200, sensor, scenario)

>>> print(results['Sensors'])
['A']
>>> print(results['Objective'])
17.3
>>> print(results['Assessment'])
  Scenario Sensor  Impact
0       S1      A     2.0
1       S2      A     3.0
2       S3   None   100.0
```

6.2 Coverage

Sensors can also be placed to maximize coverage. Coverage uses the P-median formulation and translates the impact assessment internally. The 'use_sensor_cost' and 'use_scenario_probability' flags can be used with coverage. The user can also select if sensors are placed to maximize scenario coverage or time coverage using the 'coverage_type' flag (set to 'scenario' or 'time').

Data requirements for coverage are the same as data requirements for the P-median formulation with the following exceptions:

- If 'coverage_type' is set to 'time', then the impact assessment must be a list of detection times for each sensor that detects a scenario.
- Undetected Impact is not required for each scenario.

The following example demonstrates the use of time coverage sensor placement. The results list scenario-time pairs that were detected by the sensor placement (listed as a (time, scenario) tuple). The impact value is 1 if the scenario-time pair was detected, and 0 otherwise.

```
>>> print(det_times)
  Scenario Sensor         Impact
0       S1      A     [2, 3, 4]
1       S2      A           [3]
2       S3      B  [4, 5, 6, 7]
>>> print(sensor)
  Sensor    Cost
0      A   100.0
1      B   200.0
2      C   500.0
3      D  1500.0
>>> print(scenario)
  Scenario  Undetected Impact  Probability
0       S1               48.0         0.25
1       S2              250.0         0.60
2       S3              100.0         0.15

>>> coverage = chama.optimize.Coverage(use_sensor_cost=True, coverage_type='time')
>>> results = coverage.solve(det_times, 200, sensor, scenario)

>>> print(results['Sensors'])
['B']
>>> print(results['Objective'])
0.5
>>> print(results['Assessment'])
    Scenario Sensor  Impact
0  (4, 'S3')      B     1.0
1  (5, 'S3')      B     1.0
2  (6, 'S3')      B     1.0
3  (7, 'S3')      B     1.0
4  (2, 'S1')   None     0.0
5  (3, 'S1')   None     0.0
6  (3, 'S2')   None     0.0
7  (4, 'S1')   None     0.0
```

7 Graphics

The `chama.graphics` module provides methods to help visualize results.

7.1 Signal graphics

Chama provides several functions to visualize signals described in the *Transport simulation* section (XYZ format only). Visualization is useful to verify that the signal was loaded/generated as expected, compare scenarios, and to better understand optimal sensor placement.

The convex hull of several scenarios can be generated as follows (Figure 2):

```
>>> chama.graphics.signal_convexhull(signal, scenarios=['S1', 'S2', 'S3'],␣
↪threshold=0.01)
```

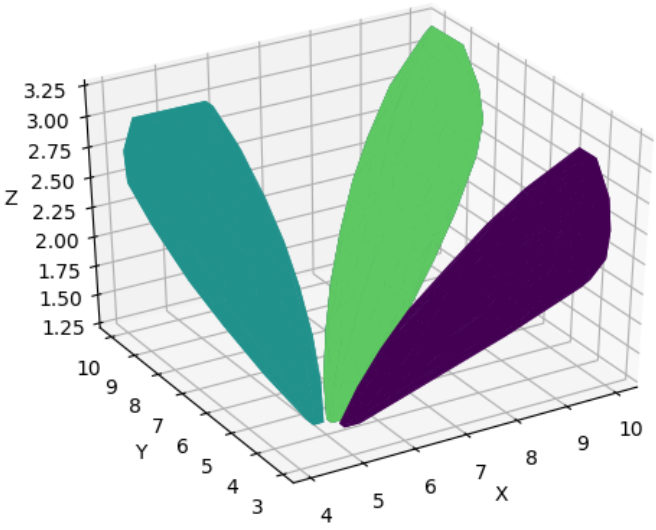


Figure 2: Convex hull plot

The cross section of a single scenarios can be generated as follows (Figure 3):

```
>>> chama.graphics.signal_xsection(signal, 'S1', threshold=0.01)
```

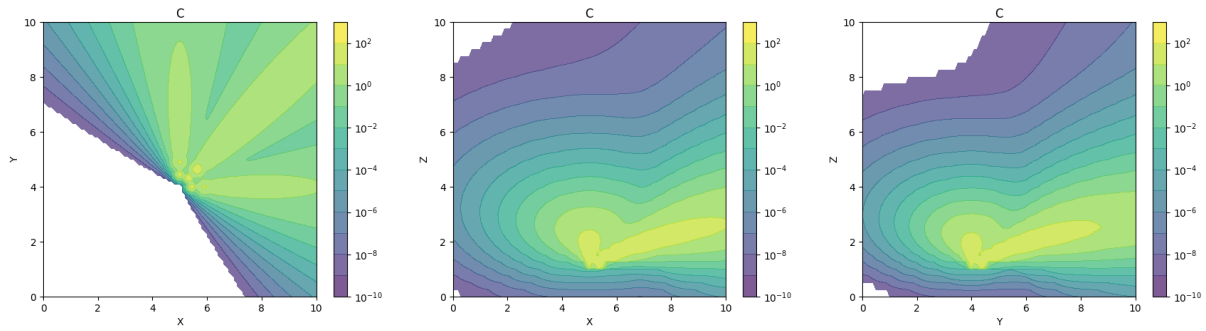


Figure 3: Cross section plot

7.2 Sensor graphics

The position of fixed and mobile sensors, described in the *Sensor technology* section, can be plotted. After grouping sensors in a dictionary, the locations can be plotted as follows (Figure 4):

```
>>> chama.graphics.sensor_locations(sensors)
```

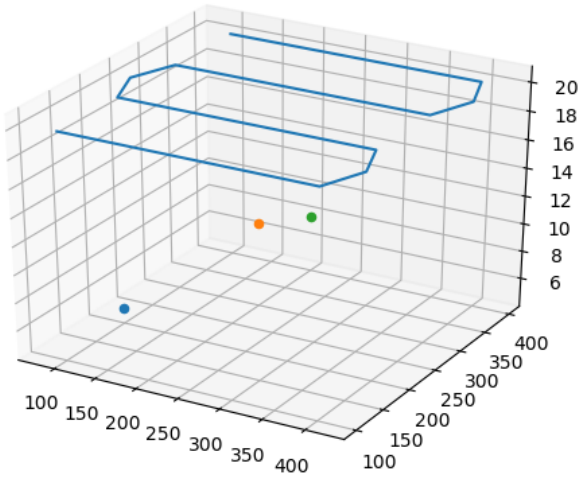


Figure 4: Mobile and stationary sensor locations plot

7.3 Tradeoff curves

After running a series of sensor placement optimizations with increasing sensor budget, a tradeoff curve can be generated using the objective value (results['Objective']). Figure 5 compares the expected time to detection (using P-median) and scenario coverage as the sensor budget increases.

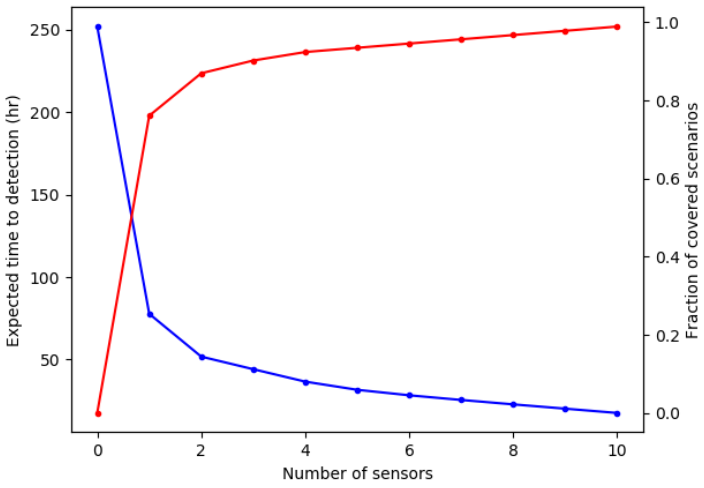


Figure 5: Optimization tradeoff curve

7.4 Scenario analysis

The impact of individual scenarios can also be analyzed for a single sensor placement using the optimization assessment (results['Assessment']). Figure 6 compares time to detection from several scenarios, given an optimal placement.

```
>>> print(results['Assessment'])
  Scenario Sensor  Impact
0       S1      A       4
1       S2      A       5
2       S3      B      10
3       S4      C       3
4       S5      A       1
>>> results['Assessment'].plot(kind='bar')
```

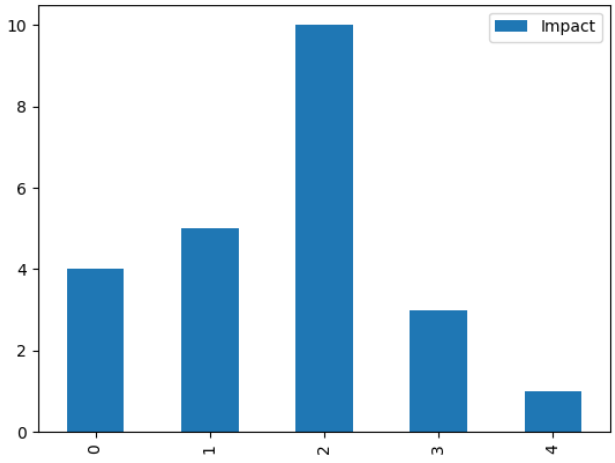


Figure 6: Scenario impact values based on optimal placement

8 Copyright and license

Chama is copyright through National Technology & Engineering Solutions of Sandia. The software is distributed under the Revised BSD License. Chama also leverages a variety of third-party software packages, which have separate licensing policies.

8.1 Copyright

8.2 Revised BSD license

9 Software development

The following section includes information on resources associated with the Chama software project, including the GitHub repository, the Python Package Index (PyPI), software tests, documentation, bug reports, feature requests, and information on contributing.

GitHub: The Chama software repository is hosted on GitHub at https://github.com/sandialabs/chama.

PyPI: The latest stable version is hosted on PyPI at https://pypi.python.org/pypi/chama.

Testing: Automated testing is run using TravisCI at https://travis-ci.org/sandialabs/chama. Test coverage statistics are collected using Coveralls at https://coveralls.io/github/sandialabs/chama. Tests can be run locally using nosetests:

```
nosetests -v --with-coverage --cover-package=chama chama
```

Documentation: Documentation is built using Read the Docs and hosted at https://chama.readthedocs.io.

Bug reports and feature requests: Bug reports and feature requests can be submitted to https://github.com/sandialabs/chama/issues. The core development team will prioritize requests.

Contributing: Software developers are expected to follow standard practices to document and test new code. Pull requests will be reviewed by the core development team. See https://github.com/sandialabs/chama/graphs/contributors for a list of contributors.

10 References

[1] Arya, S.P. (1999). Air pollution meteorology and dispersion (Vol. 6). New York: Oxford University Press.

[2] Hart, W.E., Laird, C, Watson, J.P., & Woodruff D.L. (2012). Pyomo – Optimization Modeling in Python, Volume 67 of Springer Optimization and Its Applications, Springer Science & Business Media.

[3] Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 3(9), 90-95.

[4] Klise, K.A., Hart, D.B., Moriarty, D., Bynum, M., Murray, R., Burkhardt, J., Haxton, T. (2017). Water Network Tool for Resilience (WNTR) User Manual, U.S. Environmental Protection Agency Technical Report, EPA/600/R-17/264, 47p.

[5] Legg, S.W., Benavides-Serrano, A.J., Siirola, J.D., Watson, J.P., Davis, S.G., Bratteteig, A., & Laird, C.D. (2012) A Stochastic Programming Approach for Gas Detector Placement Using CFD-Based Dispersion Simulations, Computers & Chemical Engineering, Volume 47, Pages 194-201.

[6] Makhorin, A. (2010). GNU Linear Programming Kit Reference Manual for GLPK Version 4.45, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia.

[7] McDonald, M.G., & Harbaugh, A.W. (1988). A Modular Three-dimensional Finite-Difference Groundwater Flow Model: U.S. Geological Survey Techniques of Water-Resources Investigations, Book 6, Chap. A1, 586p.

[8] McKinney W. (2013). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, 1 edition, 466P.

[9] Ravikumar, A.P., Wang, J., Brandt, A.R. (2016). Are Optical Gas Imaging Technologies Effective For Methane Leak Detection? Environmental science and technology 51 (1), 718-724.

[10] Rossman, L.A. (2000). EPANET 2 Users Manual. U. S. Environmental Protection Agency Technical Report, EPA/600/R–00/057, 200p.

[11] Scire, J., Strimaitis, D.G., & Yamartino, R.J. (2000). A User's Guide for the CALPUFF Dispersion Model (Version 5), Earth Tech, Inc.

[12] United States Environmental Protection Agency. (2004). User's Guide for the AMS/EPA Regulatory Model - AERMOD, U. S. Environmental Protection Agency Technical Report, EPA-454/B-03-001.

[13] United States Environmental Protection Agency. (2012). TEVA-SPOT Toolkit User Manual, U. S. Environmental Protection Agency Technical Report, EPA/600/R-08/041B.

[14] United States Environmental Protection Agency. (2015). Water Security Toolkit User Manual, U. S. Environmental Protection Agency Technical Report, EPA/600/R-14/338, 187p.

[15] van der Walt, S., Colbert, S.C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. Computing in Science & Engineering, 13, 22-30.

DISTRIBUTION

1	MS0735	Erik Webb	8860 (Electronic copy)
1	MS0735	Leigh Cunningham	8862 (Electronic copy)
1	MS0750	Lori Parrott	8863 (Electronic copy)
1	MS0899	Technical Library	9536 (Electronic copy)


Sandia National Laboratories