SANDIA REPORT

SAND2007-5983
Unlimited Release
Printed September 2007

LDRD Final Report: Robust Analysis of Large-scale Combinatorial Applications

William E. Hart, Robert D. Carr, Cynthia A. Phillips, Jean-Paul Watson,
Nicholas L. Benavides, Harvey Greenberg, and Todd Morrison

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

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.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent 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 the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

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@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2007-5983
Unlimited Release
Printed September 2007

LDRD Final Report: Robust Analysis of Large-scale Combinatorial Applications

William E. Hart, Robert D. Carr, Cynthia A. Phillips, and Jean-Paul Watson
Discrete Math and Complex Systems Department
Sandia National Laboratories
P.O. Box 4800
Albuquerque, NM 87185-1318

Nicolas L. Benavides
Santa Clara University
500 El Camino Real
Santa Clara, CA 95053

Harvey Greenberg and Todd Morrison
Department of Mathematics
University of Colorado, Denver
1200 Larimer St.
Denver, CO 80204

Abstract

Discrete models of large, complex systems like national infrastructures and complex logistics frameworks naturally incorporate many modeling uncertainties. Consequently, there is a clear need for optimization techniques that can robustly account for risks associated with modeling uncertainties. This report summarizes the progress of the Late-Start LDRD "Robust Analysis of Large-scale Combinatorial Applications". This project developed new heuristics for solving robust optimization models, and developed new robust optimization models for describing uncertainty scenarios.

Acknowledgments

We thank Regan Murray for collaborating on the application of robust optimization techniques to water security applications.

Contents

1 Executive Summary 9
2 Robust Optimization with CVaR 11
2.1 A CVaR Integer Programming Formulation 11
2.2 Preliminary Computational Results 14
2.3 Minimizing CVaR 15
3 A New Formulation for Minimizing Regret 19
3.1 A Minimax Regret Formulation 19
3.2 A New Minimax Regret Formulation 21
3.3 Facility Location and Sensor Placement 24
4 Enabling Technologies 25
4.1 The Pyomo Modeling Tool 25
4.2 Goal Programming in PICO 26
References 28

Appendix

A Robust Optimization Journal Article 29
B Pyomo Technical Report 30

Figures

1 Data points generated showing trade-offs between CVaR and the expected performance. 18

Tables

1 Results for minimizing expected performance with increasingly tight bounds on CVaR. Runs are limited to 3600 seconds, Gap $= \frac{\text{Incumbent} - \text{Best Bound}}{\text{Incumbent}}$ . . . 15

2 Results for minimizing expected performance with increasingly tight bounds on CVaR. Runs are limited to 1800 seconds, Gap $= \frac{\text{Incumbent} - \text{Best Bound}}{\text{Incumbent}}$ . . . 16

3 Results for minimizing weighted sum formulation with bounds on CVaR. . . 17

4 Results for minimizing CVaR with heuristic solvers. 18

5 Runtime results in seconds for computing an LP bound on CVaR. 18

1 Executive Summary

Many real-world problems are concerned with maximizing or minimizing an objective (e.g. maximizing profit, minimizing costs, or lowering of risk). *Optimization* methods are commonly used for these applications to find a best possible solution to a problem mathematically, which improves or optimizes the performance of the system. Many real-world optimization problems involve discrete decisions, such as selecting investments, allocating resources and scheduling activities. These *discrete optimization* problems arise in many application areas like infrastructure surety, military inventory and transportation logistics, production planning and scheduling, and informatics.

Discrete models of large, complex systems like national infrastructures and complex logistics frameworks naturally incorporate many modeling uncertainties. Model factors like transportation times and demands in water networks are inherently variable. Further, other information like logistical costs and infrastructure capacity limitations may only be known at a coarse, aggregate level of precision. Although such models can be optimized using average or estimated data, solutions found in this manner often fail to reflect the risks associated with these modeling uncertainties.

Consequently, there is a clear need for discrete optimization methods that can robustly account for risks associated with modeling uncertainties. So called *robust optimization* techniques find solutions that optimize a performance objective while accounting for these modeling uncertainties. Sandia's discrete optimization group has developed robust optimization methods for a variety of real-world problems, but a consistent challenge has been that existing robust optimization approaches cannot be reliably used on large-scale applications.

This report summarizes the progress of the Late-Start LDRD "Robust Analysis of Large-scale Combinatorial Applications". This project's accomplishments can be grouped into three areas:

- **Robust Optimization with CVaR**: Conditional value-at-risk (CVaR) is a risk metric that is commonly used in financial models. Discrete optimization formulations of CVaR have recently been developed for water security and facility location applications, but only small-scale problems can be practically solved with existing optimization solvers. We describe experimental analyses of CVaR problems that (a) characterize computational bottlenecks and (b) evaluate the performance of heuristic optimization solvers.

- **Minimizing Regret**: The *regret* of a decision made under uncertainty refers to the impact of not having made an optimal decision without uncertainties. Uncertainty in discrete optimization models can often be analyzed by minimizing the maximum regret. We present a new minimax regret formulation that is mathematically stronger than previous approaches.

- **Enabling Technologies**: Several software development efforts were initiated to enable the solution of robust optimization applications. A new search strategy for the PICO integer programming solver was developed to avoid over-constraining the search in risk-constrained applications. Further, a Python module was developed to flexibly model and solve the nonlinear formulations that commonly arise in discrete optimization problems.

We expect these new capabilities to directly impact Sandia's ability to address modeling uncertainties in a variety of new and ongoing efforts. For example, the following projects will leverage these capabilities in FY08:

- **Water Security (EPA)**: The EPA has been funding Sandia to develop contaminant warning systems for water distribution systems. A key element of this is the placement of sensors, which involves uncertain data. The EPA is interested in Sandia's robust optimization capabilities, and the new formulation developed in this LDRD addresses a key scalability challenge in this work: accounting for seasonal demand varitions. We do not expect this model to be used within the current SNL-EPA WFO project, but in FY08 we will leverage this when formulating a new WFO project with the EPA.

- **Scheduling Investments in Future Energy Supplies (LDRD)**: An ongoing LDRD project will leverage these robust optimization capabilities to address uncertainties in models used to plan investments our nation's energy infrastructure. Addressing uncertainties is a fundamental aspect of these models, but robust optimization is not the technical focus. However, our robust optimization results can be immediately applied to these models.

- **Aircraft Fleet Planning (LMSV)**: An ongoing Shared Vision project with Lockheed Martin will leverage the Pyomo software developed in this project to support a new application initiative. This software will be used to formulate and solve an aircraft fleet planning logistics model. In particular, Pyomo's ability to interface with aircraft design sub-models is critical to this project. Further, there is significant uncertainty in these logistics problems, and thus robust optimization techniques are particularly valuable for these planning activities.

2 Robust Optimization with CVaR

The conditional value-at-risk (CVaR) metric is a risk measure that has been widely used in the finance community. The CVaR risk measure can be applied to applications in which problem uncertainty can be characterized by a set of scenarios. For example, in infrastructure security applications, scenarios might consider possible failures of infrastructure components due to attacks. In logistics models, scenarios might characterize the time needed to transport materials.

These types of scenario-based optimization formulations are very flexible. They can integrate complex uncertainty models. For example, they allow optimization methods to be used effectively with data generated by more detailed simulation models. Further, they allow for the characterization of the impacts of uncertainties in a generic manner.

For example, we have recently used CVaR to model risk in a water security applications [13]. The goal of this model was to place sensors so as to minimize the expected impact of a contamination event (see model (SP) below). In this application, contamination impacts are considered for a variety of scenarios that are defined by the time, location and contaminant characteristics of possible contamination events. A variety of contamination impact metrics have been developed, such as minimizing population consumption of contaminated water and minimizing time to detection. Evaluation of a scenario's impact involves a hydraulic simulation in a water distribution system, including a simulation of contaminant transport in the network.

In this section we describe an integer programming (IP) model for CVaR for sensor placement. Preliminary computational results highlight challenges with solving this IP on real-world sensor placement applications. We address this challenge in several ways. First, we consider bottlenecks in the solution of the CVaR integer program (IP); specifically, we consider the runtime of the root linear programming relaxation. Although the cost of this relaxation can be reduced, the total cost of the CVaR IP remains large. Consequently, we consider the application of several IP heuristics.

2.1 A CVaR Integer Programming Formulation

Value-at-Risk (VaR) is a percentile-based metric usually defined as the maximal allowable loss within a certain confidence level $\gamma \in (0,1)$ [12]. Mathematically, suppose we have a function, $f(\vec{x},\xi)$, where $\vec{x}$ is a vector of decision variables and ξ is a vector of random variables. Then $f(\vec{x},\xi)$ is also a random variable, and we define:

$$\mathrm{VaR}(\vec{x},\gamma) = \min\{u : \Pr[f(x,\xi) \leq u] \geq \gamma\}.$$

For example, suppose $f(\vec{x},\xi) = \xi$ and $\Pr(\xi = 10) = \Pr(\xi = 100) = \frac{1}{2}$. Then,

$$\mathrm{VaR}(x,\gamma) = \begin{cases} 10 & \text{if} \quad 0 \leq \gamma < .5 \\ 100 & \text{if} \quad .5 \leq \gamma < 1. \end{cases}$$

If we think of f as some measure of risk, the chance constraint is a confidence level (γ) that the risk not exceed some level, which we minimize. (We can equivalently write $\Pr[f(\vec{x},\xi) > u] \leq 1-\gamma$, which says that we want the probability of exceeding some level (u) to be less than $1-\gamma$, say 5%. The objective is to minimize that level subject to that chance constraint.) We generally want to know what the VaR is at various values of γ, ranging from 90% to 99%.

The Conditional Value-at-Risk (CVaR) is a related metric which measures the conditional expectation of losses exceeding VaR at a given confidence level. Technically, this expectation is the *Tail Conditional Expectation* (TCE),

$$\mathrm{TCE}(\vec{x},\gamma) = \mathrm{E}\left[f(\vec{x},\xi) \,\middle|\, f(\vec{x},\xi) \geq \mathrm{VaR}(\vec{x},\gamma)\right],$$

and CVaR is linearization of TCE investigated by Uryasev and Rockafellar [11]. CVaR approximates TCE with a continuous, piecewise-linear function of γ.

To see this relation, let Ω be the set of scenarios, let ω_i the probability of realizing scenario i, and $f_i(\vec{x})$ be the value of f in scenario $i \in \Omega$. Observe that by defining an indicator variable, ρ_i, to be 1 if $f_i \geq \mathrm{VaR}$ and 0 otherwise, we can write TCE as,

$$\mathrm{TCE}(x,\gamma) = \frac{\sum_i f_i \omega_i \rho_i}{\sum_i \omega_i \rho_i}.$$

Next we define a vector of auxiliary variables $y = (y_i)$ such that

$$y_i = \max\{0, f_i - \mathrm{VaR}\}.$$

Then we can write,

$$\mathrm{TCE}(x,\gamma) = \frac{\sum_i f_i \omega_i \rho_i}{\sum_i \omega_i \rho_i} = \mathrm{VaR} + \frac{\sum_i (f_i - \mathrm{VaR}) \omega_i \rho_i}{\sum_i \omega_i \rho_i} = \mathrm{VaR} + \frac{\sum_i \omega_i y_i}{\sum_i \omega_i \rho_i}.$$

Noting that $\sum_i \omega_i \rho_i \approx \gamma$, we define CVaR as the approximation,

$$\mathrm{CVaR}(x,\gamma) = \mathrm{VaR} + \frac{1}{\gamma}\sum_i \omega_i y_i.$$

Then, since $\gamma \leq \sum_i \omega_i \rho_i$, we have,

$$\mathrm{TCE}(x,\gamma) = \mathrm{VaR} + \frac{\sum_i \omega_i y_i}{\sum_i \omega_i \rho_i} \leq \mathrm{VaR} + \frac{1}{\gamma}\sum_i \omega_i y_i = \mathrm{CVaR}(x,\gamma).$$

Graphically, CVaR holds with equality at those points where $\gamma = \sum_i \omega_i \rho_i$ (the points where the usual function steps) and joins the steps with a linear overestimate of CVaR. Thus, CVaR serves as an approximation of the bilinear form found in the formulation of TCE and is continuous in γ.

We illustrate the use of CVaR by developing an IP for minimizing the CVaR of impacts in sensor placement problem. We consider the sensor placement formulation described Berry et al. [6]:

$$
\begin{aligned}
\text{(SP)} \quad \min \quad & \sum_{a \in \mathscr{A}} \alpha_a \sum_{i \in \mathscr{L}_a} d_{ai} x_{ai} && && (1)\\
& \sum_{i \in \mathscr{L}_a} x_{ai} = 1 && \forall a \in \mathscr{A} && (2)\\
& x_{ai} \leq s_i && \forall a \in \mathscr{A}, i \in \mathscr{L}_a && (3)\\
& \sum_{i \in L} s_i \leq p && && (4)\\
& s_i \in \{0,1\} && \forall i \in L && (5)\\
& 0 \leq x_{ai} \leq 1 && \forall a \in \mathscr{A}, i \in \mathscr{L}_a && (6)
\end{aligned}
$$

This IP minimizes the expected impact of a set of contamination scenarios defined by $\mathscr{A}$. For each scenario a, $\mathscr{L}_a \subseteq L$ defines the set of locations that can be contaminated in the scenario, α_a defines the weight of the scenario, and d_{ai} defines the impact of the contamination; Berry et al. [6] consider a water security application, where typical impacts are population exposure, extent of contamination, and time to detection. The s_i variables indicate where sensors are placed in the network, subject to a budget p, and the x_{ia} variables indicate whether scenario a is witness at location i by a sensor.

A limitation of this model is that is considers only the weighted average of scenario impacts, the *expected impact*. Thus rare, but potentially catastrophic, contamination scenarios will be essentially ignored. To address these possibly disastrous extremes we need to include some measure of the risk associated with a particular solution. As a risk metric that is sensitive to large tails CVaR is well suited to this task. Hence we have formulated the sensor placement problem with restricted risk:

$$
\begin{aligned}
\text{(rrSP)} \quad \min \quad & \sum_{a \in \mathscr{A}} \alpha_a \sum_{i \in \mathscr{L}_a} d_{ai} x_{ai} && && (7)\\
& \sum_{i \in \mathscr{L}_a} x_{ai} = 1 && \forall a \in \mathscr{A} && (8)\\
& x_{ai} \leq s_i && \forall a \in \mathscr{A}, i \in \mathscr{L}_a && (9)\\
& \sum_{i \in L} s_i \leq p && && (10)\\
& s_i \in \{0,1\} && \forall i \in L && (11)\\
& 0 \leq x_{ai} \leq 1 && \forall a \in \mathscr{A}, i \in \mathscr{L}_a && (12)\\
& v + \frac{1}{\gamma} \sum_{a \in \mathscr{A}} \alpha_a y_a \leq \text{maxCVaR} && && (13)\\
& y_a \geq \sum_{i \in \mathscr{L}_a} d_{ai} x_{ai} - v && \forall a \in \mathscr{A} && (14)\\
& y_a \geq 0 && \forall a \in \mathscr{A} && (15)
\end{aligned}
$$

Alternately, we can formulate the risk adverse sensor placement problem as a goal programming problem by dropping the maximum CVaR constraint (13) and replacing the objective

function (7) with,

$$\text{(raSP)} \quad \min \quad \sum_{a \in \mathscr{A}} \alpha_a \sum_{i \in \mathscr{L}_a} d_{ai} x_{ai} + \lambda \left(v + \frac{1}{\gamma} \sum_{a \in \mathscr{A}} \alpha_a y_a \right).$$

Either formulation allows us to explore the efficient frontier of trade-offs between minimizing the expected case versus reducing our exposure to risky outliers. In the first formulation this can be done by solving the problem for a number of values, maxCVaR, assuming we know a reasonable range to work in. The second formulation avoids the need to know this range, instead we explore the efficient frontier by varying the parameter λ. Computational issues and the difficulties inherent in exploring the efficient frontier of non-convex, multiple-objective problems (such as integer programs) will probably require the use of both formulations or a combination of the two.

2.2 Preliminary Computational Results

Our preliminary computational analysis of CVaR considers a small sensor placement application for water security for which we can solve many CVaR optimization problems to optimality. We explore the efficient frontier by solving for the minimum expected value subject to an upper bound ("maxCVaR") on CVaR. Since we would like to investigate only the non-trivial points we seed our search by solving a weighted sum formulation ("Min-Both") with a very small weight ($\lambda = .01$) on CVaR. Table 1 shows results from solving a sequence of problems with an increasingly tighter bound (99% of the previous bound).

Table 2 shows results from solving for the minimum of a weighted sum, expected impact plus $\lambda \times$ CVaR. In these runs run time was limited to 30 minutes, note that for some higher values of λ this was not sufficient to find even as good an incumbent as previous runs. Also, different values for λ may correspond to the same frontier point, yet require vastly different computational effort. Further, Table 3 shows a combination approach where we solve the weighted sum formulation subject to CVaR bound. These runs clearly show that the efficient frontier is stair-stepped with some points on the frontier weakly dominated.

The data points from these experiments are plotted in Figure 1, which illustrates the extent of the frontier that we have searched. Points in the lower right-hand side of this frontier are easy to find, while points on the upper left-hand side are much more difficult.

Appendix A includes an article submitted for publication to the Journal of Infrastructure Systems. This article summarizes the use of this CVaR IP for real-world sensor placement applications; Watson, Hart and Murray [13] describes a preliminary version of this article, and a journal submission was completed as part of this LDRD. This article shows the difficulty associated with optimizing large-scale CVaR models. We were not able to solve the IP formulation for real-world sensor placement applications. Further, it was difficult to assess whether heuristic optimization methods provided near-optimal solutions. Thus,

MinExpect

CVaR	Expected	Time (sec)	Nodes	Gap (%)
57237.4*	19680.3	0 +	0	0
56664.9	19707.4	0 +	99	0
56098.2	19707.4	0 +	5	0
55537.2	19781.1	0 +	256	0
54981.9	19857.6	0 +	575	0
54432.1	19931.2	50 +	1833	0
53887.7	20044.7	135 +	3147	0
53567.6	20064.9	250	5527	0
53348.9	20209.5	400 +	11449	0
53261.0	20209.5	1079	14487	0
53174.0	20299.0	1922	22117	0
53000.0	20486.5	3646 +	55747	1.50
52815.4	20764.5	3620 +	52505	2.44

Table 1. Results for minimizing expected performance with increasingly tight bounds on CVaR. Runs are limited to 3600 seconds, Gap $= \frac{\text{Incumbent} - \text{Best Bound}}{\text{Incumbent}}$

the effective application of this CVaR formulation to large discrete problems remains a challenge.

2.3 Minimizing CVaR

From our previous results, a clear challenge is the analysis of IP models with highly constrained CVaR. In particular, simply minimizing CVaR can be a very challenging problem, even for small distribution networks. IP solvers have not proven effective at minimizing CVaR because the LP bounds used in this search process are very weak. Consequently, we have focused on the application of heuristic optimizers, which work quickly but do not guarantee that an optimal is found.

In previous work, we have developed a GRASP heuristic for CVaR [6], and in this project we contrast this heuristic with two new IP solvers: a feasibility pump heuristic, and a fractional decomposition tree (FDT) heuristic. The feasibility pump heuristic is a recently developed strategy for quickly finding solutions to general integer programs. This heuristic starts with the optimal LP solution, and then solves a series of LP subproblems that attempt to drive this solution towards a feasible discrete solution. This heuristic has recently been integrated into the PICO IP solver, where it generates solutions used to prune the branch-and-bound tree used during search.

MinBoth

λ	CVaR	Expected	Time (sec)	Nodes	Gap (%)
0.05	55826.7	19707.4	12.9	4	0
0.10	55826.7	19707.4	46.2	277	0
0.20	54091.7	19951.4	235.9	2287	0
0.25	53567.6	20064.9	527.1	6239	0
0.30	53567.6	20064.9	1471.4	19751	0
0.40	53370.8	20186.4	1818.1	18894	1.32
0.50	53126.3	20429.8	1814.1	12113	2.83
1.00	51762.0	21266.7	1812.3	9974	5.59
2.00	51627.4	21586.9	1810.9	9115	6.33
5.00	52387.7	21447.2	1816.9	16080	8.52
10.00	52154.4	23753.9	1821.1	21791	10.75

Table 2. Results for minimizing expected performance with increasingly tight bounds on CVaR. Runs are limited to 1800 seconds, Gap $= \frac{\text{Incumbent} - \text{Best Bound}}{\text{Incumbent}}$

FDT starts with the optimal LP solution, which is often not discrete. It then considers a series of decompositions of the fractional solution, into a convex combination of two solutions that are integral in one-or-more variables. Thus FDT iteratively fixes discrete solutions, but in a manner that is guided by the LP fractional solution. FDT has been recently developed by Carr and Phillips [7], and our application of FDT to CVaR is one of the first evaluations of this heuristic on real-world applications. An FDT implementation for minimizing CVaR was implemented in the AMPL modeling language.

Table 4 summarizes the result of these heuristics on a distribution network with 97 junctions. For each junction, we considered the impact of contamination events simulated for each hour of the day. Four different impact metrics were considered: extent of contamination (ec) in the network, mass of consumed (mc) of contaminant by demand nodes, time to detection (td) and volume consumed (vc) of contaminated water by demand nodes.

The GRASP and feasibility pump results were generated within a few minutes. The FDT heuristic generates a series of solutions; the first solution is generated within a few minutes, and generating all subsequent solutions required up to an hour. These results indicate that feasibility pump and FDT are not an improvement over the GRASP heuristic, either in final solution value or in terms of the required runtime.

A clear limitation of heuristic methods is that they fail to provide a confidence bound in the final solution. One strategy for providing a bound is to compute the linear relaxation of the IP formulation, which relaxes the integrality constraints. The values of the LP relaxations are provided in Table 4. These relaxations are not particularly tight; the best incumbent solutions are substantially higher than these values, which further illustrates the

λ	maxCVaR	53348.9	53261.0	53174.0
0.3	CVaR	53205.3	53205.3	53053.5
	Expected	20209.5	20209.5	20299.0
	Time	1598.7	2361.8	2412.3
	Found at	14493	18329	29319
	Nodes	19139	33837	31316
	Gap (%)	0.00	0.00	0.00
0.35	CVaR	53205.3	53205.3	53053.5
	Expected	20209.5	20209.5	20299.0
	Time	2956.9	3571.7	7286.4
	Found at	24367	35379	98090
	Nodes	42451	53938	101632
	Gap (%)	0.00	0.00	0.18
0.4	CVaR	53205.3	53205.3	53053.5
	Expected	20209.5	20209.5	20299.0
	Time	3025.4	4439.5	4364.7
	Found at	12276	33579	55205
	Nodes	37535	54935	57537
	Gap (%)	0.00	0.00	0.00

Table 3. Results for minimizing weighted sum formulation with bounds on CVaR.

difficulty of optimizing CVaR with IP methods. Table 5 shows the run time for computing the LP relaxations with several different LP solvers. The CPLEX barrier solver is clearly faster than the CLP and CPLEX dual solvers.

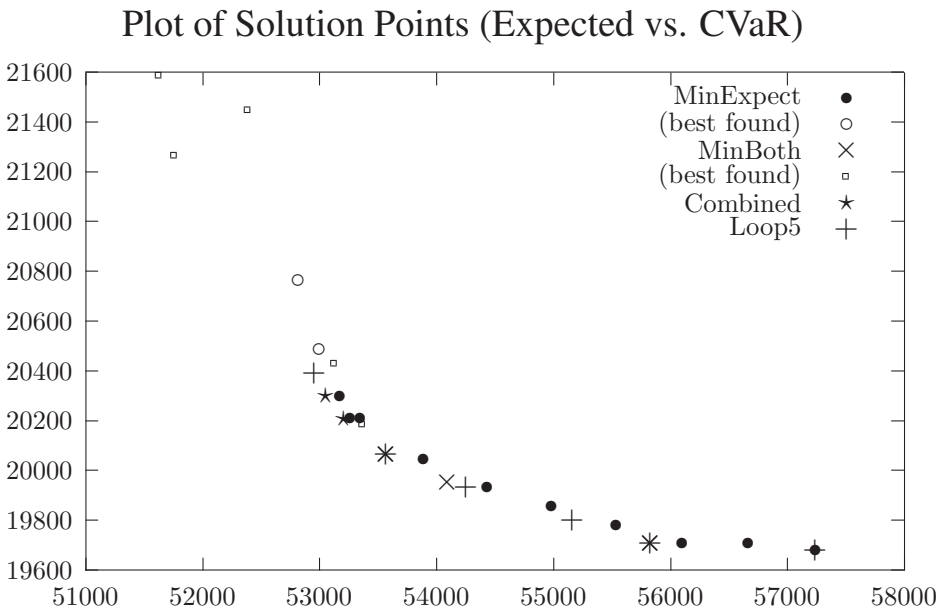

Figure 1. Data points generated showing trade-offs between CVaR and the expected performance.

Solution	ec	mc	td	vc
LP Lower Bound	16436	89709	1583	170598
GRASP	26502	144271	2555	228312
Feasibility Pump	376413	144544	2874	364921
FDT First	32353	144278	2556	1342897
FDT Best	32353	144278	2556	1051129

Table 4. Results for minimizing CVaR with heuristic solvers.

Solver	ec	mc	td	vc
CLP Dual	17.43	13.85	19.98	25.78
CLEX Dual	23.73	23.18	11.48	30.29
CPLEX Barrier	5.73	4.17	5.58	7.64

Table 5. Runtime results in seconds for computing an LP bound on CVaR.

3 A New Formulation for Minimizing Regret

The *regret* of a decision made under uncertainty refers to the impact of not having made an optimal decision without uncertainties. For example, consider the context of selecting a facility location to meet "customer" demand (e.g. locating a fire station or waste dump). Facility demands are uncertain, and these uncertainties can often be characterized as a set of potential demand scenarios. In practice, one or more of these scenarios is likely to dominate future usage of the facility, but further information is unavailable when a decision maker selects the facility location.

A minimax regret formulation minimizes the regret of a decision across all scenarios. Here, regret can be characterized as the difference between a solution and the solution value optimized for a particular scenario:

$$f_i(x) - f_i^*$$

so the canonical minimax formulation is

$$\min_x \max_i f_i(x) - f_i^*.$$

This is also called the worst-case regret, since we are minimizing the worst regret overall.

Chen et al. [8] consider a minimax regret model that minimizes the worst regret over the best $100\alpha\%$ (say 95%) of the scenarios. This is better than minimizing the worst case regret when one does not want an answer that is dominated by few scenarios (which may occur with low probability). For example, airports should not cater only to Thanksgiving and Christmas travel, but a worst-case regret formulation could do just that.

The following sections critique this model and present an alternative formulation that can be used to optimize this modified regret formulation. In particular, this reformulation is motivated by the fact that the technique described by Chen et al. [8] is not practical for large facility location applications. The final section below discusses the relationship between this facility location model and the water security application discussed above.

3.1 A Minimax Regret Formulation

The original idea behind the facility location problem is that there is a network of customer node locations with a demand at each node and potential sites for p facilities that service these demands at a cost that increases with the distance from the facility to the customer. The problem then is to place the facilities so that the total demand is met at minimum cost.

We start with demand nodes i numbered from 1 to m so that the first n nodes (from 1 to n) are the potential sites to place a facility, of which p of these sites will be chosen for putting facilities. We are given scenarios 1 to K. For scenario k we specify a demand h_{ik} for each customer i and distance d_{ijk} between each customer i and potential facility location j. We assign a probability q_k that scenario k will occur and the minimum cost $\hat{V}_k$ for satisfying

the total demand given one knows a head of time that scenario k will occur; $\hat{V}_k$ is obtained by solving a facility location problem for scenario k alone.

The robust model proposed by Chen et al. [8] finds set of facility locations that minimize a regret measure. So, if x_j for $j = 1..n$ are binary variables that indicate where we will put our facilities and $y_{i,j,k}$ are binary variables that are 1 when customer i gets serviced by facility j under scenario k, the total cost of meeting the demand for scenario k is

$$\sum_{i=1}^{m} \sum_{j=1}^{n} h_{ik} d_{ijk} y_{ijk}.$$

Since the lowest possible cost would be $\hat{V}_k$, there is a regret of R_k from scenario k given by

$$R_k = \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ik} d_{ijk} y_{ijk} - \hat{V}_k.$$

We could minimize a weighted sum of regrets or a worst-case regret, but in both cases outlier scenarios can significantly skew our solution. A worst-case regret may only be relevant for a small number of extreme scenarios, and a weighted sum can be similarly skewed by large outlier values. Although we would want a solution to work well in principle for all scenarios, that fact that we have uncertainties in the relevance of these scenarios motivates the decision maker to ignore these extreme scenarios for the analysis.

Our robust measure based on regrets is to minimize the worst regret in the best 95% of the outcomes. If we have a binary variable z_k indicating whether scenario k is in the best $100\alpha\%$ of the regrets, then we will minimize the maximum regret W where for each k we have the constraint

$$W + R_k(1 - z_k) \geq R_k,$$

which makes W larger than any regret R_k in the best $100\alpha\%$ (that is for which $z_k = 1$). Notice that these constraints are designed to say nothing we didn't already know when k is not one of the selected scenarios ($z_k = 0$), but say exactly what we want when k is a selected scenario. To ensure that the z variables are set correctly, we need the constraint

$$\sum_{k=1}^{K} q_k z_k \geq \alpha.$$

Unfortunately, the constraints that give lower bounds for W are non-linear, so we cannot solve this formulation with an integer programming solver in a standard manner. Chen et al. [8] resolve this by guessing the constants m_k to be as close to but bigger than the actual regrets R_k as possible. Then, the constraints

$$W + m_k(1 - z_k) \geq R_k$$

can be used to bound W. The problem with this approach is that if these guesses for m_k are off, one may have to make new guesses and solve the IP all over again.

Finally, we have the normal facility location constraints for the x variables (indicating facilities) and the y variables (indicating which facility services each customer). Since we are placing p facilities, we have

$$\sum_{j=1}^{n} x_j = p.$$

Since in each scenario k, only one facility services any customer i, we have

$$\sum_{j=1}^{n} y_{ijk} = 1 \quad \forall i \in \{1,..,m\} \forall k \in \{1,..,K\}.$$

Since a facility cannot service a customer if it were never built, we have

$$y_{ijk} \leq x_j \quad \forall j \in \{1,..,n\} \forall i \in \{1,..,m\} \forall k \in \{1,..,K\}.$$

As was stated earlier, our objective is to minimize W subject to the constraints of this section.

3.2 A New Minimax Regret Formulation

We have come up with several ideas for improving the formulation discussed in the previous section. The first idea is to turn the variable W into a constant by guessing its value to be some W^*. We will soon see that this makes the cost of a single scenario easier to model as a linear function, and besides we had to make guesses in the previous IP as well. In keeping with our robust modeling ideas, we should be able to model a truncated cost of a scenario k that is its actual cost if $z_k = 1$, but only $W^* + \hat{V}_k$ if k were an outlier ($z_k = 0$). Then, the cost of scenario k can be given by an almost linear function

$$(W^* + \hat{V}_k)(1 - z_k) + (R_k + \hat{V}_k) z_k. \tag{16}$$

The only non-linearity is the product $R_k z_k$, but we will explain later that this product can be closely approximated by linear variables $\overline{F}_k$, turning the single scenrio cost into the linear function

$$(W^* + \hat{V}_k)(1 - z_k) + \overline{F}_k + \hat{V}_k z_k.$$

Going back to equation (16), notice that the cost is a convex combination of the truncated cost $W^* + \hat{V}_k$ and the actual cost $R_k + \hat{V}_k$, with convex multipliers $1 - z_k$ and z_k respectively. Then, z_k takes on a value of either 0 or 1, depending on which of the actual and truncated costs were smaller since we wish to minimize cost.

We show the basic modeling idea behind the variables F_k that are used to determine $\overline{F}_k$, and are the product of the cost (optimum single scenario plus regret) times z_k.

$$F_k = (\sum_{i=1}^{m} \sum_{j=1}^{n} h_{ik} d_{ijk} y_{ijk}) z_k.$$

Now, we can impose constraints that bound the variables F_k:

$$\begin{array}{rcccl} & & F_k & = & \overline{F}_k + \hat{V}_k z_k, \\ \hat{V}_k z_k & \leq & F_k & \leq & (W^* + \hat{V}_k) z_k. \end{array}$$

These constraints ensure that if the cost exceeds the threshold, that is

$$F_k / z_k := \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ik} d_{ijk} y_{ijk} > W^* + \hat{V}_k,$$

then z_k and F_k would be set to 0, which one can afford to do up to $1-\alpha$ of the time, so that F_k would not exceed $(W^* + \hat{V}_k) z_k$.

Our next idea is to define scaled versions of the LP relaxation for the facility location formulation so that we could effectively model F_k and $\overline{F}_k$, the versions of the cost and regret of the solution scaled by z_k, for each scenario k. To achieve this, we create variables t_{jk} and v_{ijk} that we wish to satisfy

$$\begin{array}{rcl} v_{ijk} & := & y_{ijk} z_k \\ t_{jk} & := & x_j z_k. \end{array}$$

The usual LP relaxation for facility location is

$$\begin{array}{rcl} \sum_{j=1}^{n} x_j & = & p \quad 0 \leq x \leq 1 \\ \sum_{j=1}^{n} y_{ij} & = & 1 \quad \forall i \in \{1,..,m\}, \quad y \geq 0 \\ y_{ij} & \leq & x_j \quad \forall i \in \{1,..,m\} \forall j \in \{1,..,n\}, \\ \text{cost} & = & \sum_{i=1}^{m} \sum_{j=1}^{n} h_i d_{ij} y_{ij}. \end{array} \tag{17}$$

Our constraints to scale this LP by z_k are:

$$\begin{array}{rcl} \sum_{j=1}^{n} t_{jk} & = & p z_k \quad 0 \leq t_{jk} \leq z_k \forall j \in \{1,..,n\}, \\ \sum_{j=1}^{n} v_{ijk} & = & z_k \quad \forall i \in \{1,..,m\}, \quad v \geq 0 \\ v_{ijk} & \leq & t_{jk} \quad \forall i \in \{1,..,m\} \forall j \in \{1,..,n\}, \\ F_k & = & \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ik} d_{ijk} v_{ijk}. \end{array} \tag{18}$$

If we take the above formulation and divide each of t, v, and F by z_k, we get the usual LP relaxation for facility location, which means that these scaled models create no additional error compared with the LP relaxation other than that from relaxing the binary variables of the problem.

Our third idea is to enforce t to be z_k multiplied by the same x vector for all k while allowing v to be z_k multiplied by a different y_{ijk} vector for each scenario k. This makes sense since we want to make a placement of facilities that does not depend on scenario while which facility services a customer could depend on the scenario. In fact, we do not use a k subscript for the x variables, but do use such a subscript for the y variables. This modeling distinction leads us to form constraints analogous to the single constraint

$$\sum_{k=1}^{K} q_k z_k \geq \alpha. \tag{19}$$

We may now have a constraint for each $j \in \{1,..,n\}$ stating:

$$\sum_{k=1}^{K} q_k t_{jk} \geq \alpha x_j. \tag{20}$$

Also, we can form another constraint analogous to

$$0 \leq t_{jk} \leq z_k, \tag{21}$$

based on the idea that $x_j - t_{jk} = x_j(1-z_k)$, so we can multiply $0 \leq x \leq 1$ by $1-z_k$ as well as by z_k. Hence, we get

$$0 \leq x_j - t_{jk} \leq 1 - z_k. \tag{22}$$

These are similar to the constraints

$$0 \leq y_{ijk} - v_{ijk} \leq 1 - z_k, \tag{23}$$

except that we take advantage of x not depending on the scenario k.

Our robust model in its entirety, except that the objective function is left out, is as follows:

$$\begin{array}{rcl}
F_k & = & \sum_{i=1}^{m}\sum_{j=1}^{n} h_{ik} d_{ijk} v_{ijk} \quad \forall k \in [K] \\
\hat{V}_k z_k & \leq & F_k \leq (W^* + \hat{V}_k) z_k \quad \forall k \in [K] \\
\sum_{j=1}^{n} x_j & = & p \quad 0 \leq x \leq 1 \\
\sum_{j=1}^{n} y_{ijk} & = & 1 \quad \forall i \in [m] \forall k \in [K] \quad y \geq 0 \\
y_{ijk} & \leq & x_j \quad \forall i \in [m] \forall j \in [n] \forall k \in [K] \\
\sum_{j=1}^{n} t_{jk} & = & p z_k \quad 0 \leq t_{jk} \leq z_k \forall j \in [n] \forall k \in [K] \\
\sum_{j=1}^{n} v_{ijk} & = & z_k \quad \forall i \in [m] \forall k \in [K] \quad v \geq 0 \\
v_{ijk} & \leq & t_{jk} \quad \forall i \in [m] \forall j \in [n] \forall k \in [K] \\
\sum_{k=1}^{K} q_k z_k & \geq & \alpha \\
\sum_{k=1}^{K} q_k t_{jk} & \geq & \alpha x_j \quad \forall j \in [n] \\
0 & \leq & x_j - t_{jk} \leq 1 - z_k \quad \forall j \in [n] \forall k \in [K] \\
0 & \leq & y_{ijk} - v_{ijk} \leq 1 - z_k \quad \forall i \in [m] \forall j \in [n] \forall k \in [K] \\
x, y, z, t, v & & \text{integer.}
\end{array} \tag{24}$$

As for the objective function, we have choices. One idea is for this IP to simply be a feasibility problem with no objective function. Another idea is to add up the cost of each scenario k truncated by $W^* + \hat{V}_k$, and minimize. Thus an objective function could be

$$\text{minimize} \sum_{k=1}^{K} ((W^* + \hat{V}_k)(1 - z_k) + F_k).$$

One can see by examining this IP model when the z variables all have $0,1$ values that its LP relaxation is as tight as that of the facility location problem for each scenario when this integality condition is satisfied. This indicates that ours is a better LP relaxation than that of the previous section. This also indicates that branching on the z variables is particularly important when solving our robust IP with branch-and-bound.

3.3 Facility Location and Sensor Placement

The sensor placement model (SP) discussed in Section 2 is closely related to the standard p-median formulation used for facility location. There is some additional structure that can be exploited in (SP), but otherwise it uses the same set of constraints. However, the CVaR formulation and our minimax regret formulation address different aspect of modeling uncertainties in sensor placement applications.

The CVaR model was developed to characterize the risk associated with different contamination events. In general, we wish to optimize expected performance, while constraining such risk to acceptable level. But when drawing a correspondence with facility location, the CVaR model considers only a single *scenario*; the different contamination events correspond to different demands on a facility.

To better understand this correspondence, consider the placement of sensors to protect against contamination events at different seasons of the year. Water usage patterns will be quite different between summer and winter, and thus contamination events could propagate in very different manners and have different consequences. However, for each season there is a set of possible contamination events that need to be considered, for different locations and times of day for contamination. Thus, the seasons correspond to the scenarios that we have considered for facility location.

This generalization of the sensor placement problem addresses one of the key limitations of existing sensor placement formulations. There are a large number of possible scenarios that account for different conditions in the network and different characteristics of contamination events. However, existing approaches lump all of the contamination events in these scenarios into one set of events. As we noted earlier, this could lead to sensor placement designs that are skewed towards particular contamination events in particular scenarios.

4 Enabling Technologies

Two enabling technologies were developed as part of our research efforts to facilitate the solution of robust optimization applications. A new modeling tool, Pyomo, was developed to provide a more flexible environment for modeling and solving complex formulations like robust optimization problems. Also, we developed a new search strategy for the PICO integer programming solver that can manage constraint violations in a flexible manner and cache nearly feasible solutions.

4.1 The Pyomo Modeling Tool

Appendix B includes a technical report that describes the Python Optimization Modeling Objects (Pyomo) package. Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Pyomo provides a capability that is commonly associated with algebraic modeling languages like AMPL and GAMS. However, Pyomo can leverage Python's programming environment to support the development of complex models and optimization solvers in the same modeling environment.

Algebraic Modeling Languages (AMLs) are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [10]. AMLs like AIMMS [1], AMPL [2, 9] and GAMS [4] have programming languages with an intuitive mathematical syntax that supports concepts like sparse sets, indices, and algebraic expressions. AMLs provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is essential for real-world problems that can involve thousands of constraints and variables.

An alternative strategy for modeling mathematical problems is to use a standard programming language in conjunction with a software library that uses object-oriented design to support similar mathematical concepts. Although these modeling libraries sacrifice the intuitive mathematical syntax of an AML, they allow the user to leverage the greater flexibility of standard programming languages. For example, modeling libraries like FLOPC++ [3] and OPL [5] enable the solution of large, complex problems within a user-defined application.

Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Like other modeling libraries, Pyomo can generate problem instances and apply optimization solvers with a fully expressive programming language. Further, Python is a noncommercial language with a very large user community, which will ensure robust support for this language on a wide range of compute platforms.

Python is a powerful dynamic programming language that has a very clear, readable syntax and intuitive object orientation. Python's clean syntax allows Pyomo to express

mathematical concepts in a reasonably intuitive manner. Further, Pyomo can be used within an interactive Python shell, thereby allowing a user to interactively interrogate Pyomo-based models. Thus, Pyomo has many of the advantages of both AML interfaces and modeling libraries.

Pyomo was developed as part of this project to facilitate the development of heuristic optimizers for complex applications like robust optimization problems. Specifically, our goal was to develop heuristics like FDT in Python using Pyomo's modeling objects. Unfortunately, this goal was not realized due to time constraints; instead, we implemented FDT with a rather awkward AMPL model. However, our prototype of Pyomo can be used to model and solve simple integer programming applications using Sandia's PICO IP solver. We expect Pyomo to mature as we use it for applications, and that it will play a key role in the development of new applications. In FY08, we plan to use Pyomo to analyze aircraft fleet planning applications under Lockheed Martin Shared Vision funding, including robust planning models. We currently plan to release Pyomo under an open-source license to encourage its use by external collaborators.

4.2 Goal Programming in PICO

In practice, satisfying a risk constraint exactly in a robust optimization formulation is less crucial than finding an effective compromise between the optimization objective and the performance risk. Thus, a risk constraint is better described as a *goal* that we want to meet, and risk-constrained robust optimization formulations can be effectively cast as *goal programming* models.

Solution of goal programming models for robust optimization differs from standard discrete optimization in at least two important ways. First, the outcome of robust optimization is a set of solutions that represent trade-offs between the optimization objective and risk. Thus, the optimizer needs to maintain this set of solutions, and filter out solutions that are dominated by other solutions (i.e. they are not better in either the objective or risk value than at least one other solution). This is an example of a bi-criteria optimization problem, and standard sorting techniques can be used to maintain a set of undominated solutions.

Second, the search process needs to be adapted to explicitly recognize goals. Standard discrete optimization techniques do not allow the search to focus on infeasible solutions; in fact, the efficiency of a discrete optimization solver is often related to how well it eliminates infeasible solutions. When considering goal constraints, we need to allow for infeasible solutions. This can be done by biasing search towards solutions that meet our goals. This is a natural extension of many heuristic solvers, which simply augment the objective with a penalty associated with how much the goal constraint is violated.

Support for goal constraints is being added to Sandia's PICO integer programming solver. The heuristic solvers that PICO supports for integer programming formulations can recognize goals and treat them appropriately, and PICO maintains a pool of solutions

that represent different trade-offs between the optimization objective and these goal values. This capability will be included in an forthcoming release of PICO (planned for fall of 2007).

References

[1] *AIMMS home page*.

[2] *AMPL home page*.

[3] *FLOPC++ home page*.

[4] *GAMS home page*.

[5] *OPL home page*.

[6] J. BERRY, W. E. HART, C. E. PHILLIPS, J. G. UBER, AND J.-P. WATSON, *Sensor placement in municiple water networks with temporal integer programming models*, J. Water Resources Planning and Management, 132 (2006), pp. 218–224.

[7] R. D. CARR AND C. A. PHILLIPS, *Fractional decomposition trees: Finding feasible solutions to integer programs with bounded integrality gaps*, Tech. Report SAND 2006-7947, Sandia National Laboratories, 2007.

[8] G. CHEN, M. S. DASKIN, Z.-J. SHEN, AND S. URYASEV, *The alpha-reliable mean-excess regret model for stochastic facility location modeling*, Naval Research Logistics. (to appear).

[9] R. FOURER, D. M. GAY, AND B. W. KERNIGHAN, *AMPL: A Modeling Language for Mathematical Programming, 2nd Ed.*, Brooks/Cole–Thomson Learning, Pacific Grove, CA, 2003.

[10] J. KALLRATH, *Modeling Languages in Mathematical Optimization*, Kluwer Academic Publishers, 2004.

[11] R. T. ROCKAFELLAR AND S. URYASEV, *Conditional value-at-risk for general loss distributions*, Journal of Banking and Finance, 26 (2002), pp. 1443–1471.

[12] N. TOPALOGLOU, H. VLADIMIROU, AND S. ZENIOS, *CVaR models with selective hedging for international asset allocation*, Journal of Banking and Finance, 26 (2002), pp. 1535–1561.

[13] 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 Robust Optimization Journal Article

Formulation and Optimization of Robust Sensor Placement Problems for Drinking Water Contamination Warning Systems

Jean-Paul Watson* Regan Murray† William E. Hart‡

Abstract

The sensor placement problem in contamination warning system design for water distribution networks involves maximizing the protection level afforded by limited numbers of sensors, typically quantified as the expected impact of a contamination event; the issue of how to mitigate against high-impact events is either handled implicitly or ignored entirely. Consequently, expected-case sensor placements run the risk of failing to protect against high-impact, 9/11-style attacks. In contrast, robust sensor placements address this concern by focusing strictly on high-impact events and placing sensors to minimize the impact of these events. We introduce several robust variations of the sensor placement problem, distinguished by how they quantify the potential damage due to high-impact events. We explore the nature of robust versus expected-case sensor placements on three real-world, large-scale networks. We find that robust sensor placements can yield large reductions in the number and magnitude of high-impact events, for modest increases in expected impact. The resulting ability to trade-off between robust and expected-case impacts is a key, unexplored dimension in contamination warning system design.

Keywords

Sensor Placement, Contamination Warning System Design, Robust Optimization, Drinking Water, Homeland Security.

*Discrete Algorithms and Math Department, Sandia National Laboratories, P.O. Box 5800, MS 1318, Albuquerque, NM 87185; E-Mail: jwatson@sandia.gov

†U.S. Environmental Protection Agency, 26 W. Martin Luther King Drive, MS 163, Cincinnati, OH 45268; E-Mail: Murray.Regan@epamail.epa.gov

‡Discrete Algorithms and Math Department, Sandia National Laboratories, P.O. Box 5800, MS 1318, Albuquerque, NM, 87185; E-Mail: wehart@sandia.gov

1 Introduction

Contamination warning systems (CWSs) have been proposed as a promising approach for detecting contamination events in drinking water distribution systems. The goal of a CWS is to detect contamination events early enough to allow for effective public health and/or water utility intervention to limit potential public health or economic impacts. 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 drinking water community has proposed that CWSs be designed to maximize the number of contaminants that can be detected in drinking water distribution systems by combining online sensors with public health surveillance systems, physical security monitoring, customer complaint surveillance, and routine sampling programs (USEPA, 2005).

Computational techniques for placing sensors to support the design of CWSs for municipal water distribution networks have received significant attention from researchers and practitioners over the last ten years (Kessler et al., 1998; Ostfeld and Salomons, 2004; Berry et al., 2005a, 2006b). Without exception, these techniques attempt to either minimize the expected impact of a contamination event (e.g., in terms of the number of people sickened or the volume of contaminated water consumed) or maximize the proportion of contamination events that are ultimately detected, independent of impact. Recently, Berry et al. (2006b) showed that both objectives can be formulated in terms of a single optimization model, illustrating that the proportion of events detected can be viewed as an expected impact, and vice versa. In this unified optimization model, contamination event probabilities are either assumed to be uniform, or are estimated based on factors such as the difficulty of accessing a particular component of a distribution network. Given a broad range of possible contamination events, sensor placement techniques then attempt to minimize the probability-weighted sum of contamination event impact, i.e., the expected impact. The most advanced techniques currently available can successfully generate optimal sensor placements to very large (e.g., 10,000+ junction) distribution networks for very large numbers (e.g., 50,000+) of possible contamination events, in modest run-times on a

modern computing workstation (Berry et al., 2006b). Consequently, the basic sensor placement problem for CWS design is largely solved for most distribution networks (although practical implementation issues, such as reductions in the run-time memory requirements to facilitate deployment on low-end computing platforms, are still under active investigation), and the research emphasis has moved toward the integration of more realistic modeling assumptions such as sensor failures (Berry et al., 2006a), site specific installation costs and accessibility considerations (Berry et al., 2005b), significantly larger numbers of possible contamination events (Berry et al., 2007), and solution robustness in the face of data uncertainties (Carr et al., 2006).

One currently unexplored, but – we argue – critical aspect of the sensor placement problem involves variants in which the design objective is not minimization of the expected impact, but rather minimization of the worst-case impact or other "robust" measures that focus strictly on high-consequence contamination events. The lack of research into these alternative problems is perhaps counterintuitive in a post-9/11 environment. One explanation is that most environmental problems have required a focus on mitigating all risks to human health, and not just associated with those extremely high-impact events. Yet, robust sensor placement is of interest in practice. In our experience working with various US water municipalities, a common reaction when discussing the basic sensor placement problem is "Why not *only* concentrate on high-impact contamination events?" Additional motivation for pursuing robust sensor placement problems stems from the observation that sensor placements that minimize expected impacts can permit numerous high-impact contamination events (e.g., as discussed below in Section 2). Further, accurate estimation of event probabilities is notoriously difficult, allowing for unintended de-emphasis of high-impact events.

In this paper, we introduce a number of robust measures of sensor placement performance, drawing heavily from existing literature on robust optimization from the financial community. Using a variety of optimization techniques, we construct sensor placements that minimize these robust impact measures on three real-world water distribution networks. We find that sensor placements designed to minimize the expected impact admit – without exception – a non-trivial number of very high-impact contamination events. These high-impact events can be mitigated with robust sensor placements, e.g., we observe that significant reductions in the worst-case impact are possible. These reductions come at the necessary expense of an increase in the mean

impact of a contamination event. However, by exploring alternative robust sensor placements, the increase in mean impact can be minimized. We identify a number of interesting trade-offs between expected-case and robust performance measures. Additionally, we observe that the different robust performance measures we consider do not lead to similar sensor placements. Thus, it is important for decision-makers to understand robust sensor placement to develop effective CWS designs.

The remainder of this paper is organized as follows. We begin in Section 2 with a motivating example to illustrate differences in the characteristics of sensor placements that are optimal with respect to expected-case and worst-case performance. Various robust impact measures are then introduced in Section 3. Section 4 details the test networks, contamination events, sensor placement problems, and computational techniques that we use in the analysis discussed in Section 5; the latter details quantitative and qualitative differences between expected-case and robust sensor placements. We defer discussion of the specific computational characteristics of the techniques used in our analysis to Section 6, which additionally addresses the computational difficulty of robust sensor placement problems. Finally, we conclude in Section 7 with a discussion of the implications of our results.

2 Motivating Example

To concretely illustrate the relative trade-offs that are possible between expected-case and robust sensor placements, we begin with an example from a real-world water distribution network. The network is simply denoted Network2; this and other test networks are described in Section 4. Using the experimental methodology and algorithms presented below, we determine two distinct sensor placements for Network2 – given a budget of 20 sensors – that minimize the expected-case and worst-case impact of a contamination event. The precise details of the contamination events are documented in Section 4; impact is quantified as the number of people sickened by a contamination event (Murray et al., 2006).

Histograms of the impacts of various contamination events *given* the expected-case and worst-case sensor placements are shown in Figure 1; the data represent contaminant injections at each network node, for a total of approximately 1,600 events. The distribution of impacts

under the expected-case sensor placement, as shown in the left side of Figure 1, has mean and worst-case impacts of 685 and 4,902 individuals, respectively. The distribution exhibits a feature of sensor placements that minimize the expected-case: the presence of a substantial number of contamination events that yield impacts over seven times greater than that of the mean. Specifically, eight contamination events yield impacts greater than 4,000 individuals sickened, while an additional six contamination events yields impacts between 3,500 and 4,000 individuals sickened.

Next, we consider the distribution of impacts given a sensor placement that minimizes the worst-case impact of a contamination event, as shown in the right side of Figure 1. Relative to the expected-case distribution, we immediately observe a significant reduction in the number of very high-impact contamination events. In particular, the highest-impact event sickens 3,490 individuals, in contrast to 4,902 individuals under the expected-case sensor placement; the 14 highest-impact events in the expected-case placement are mitigated by a sensor placement that minimizes the worst case. However, as is expected, the mitigation of high-impact events increases the frequency of small-to-moderate impact events. The worst-case sensor placement yields a mean impact of 882 individuals sickened, representing a 29% increase relative to the expected-case sensor placement. Even more dramatic growth is observed in the upper bound of the third impact quartile, from 1,011 under the expected-case sensor placement to 1,445 under the worst-case sensor placement (representing a 43% increase). For decision-makers in CWS design, this raises the question: Is a large (in this case 29%) reduction in the worst-case impact worth a correspondingly large increase in the expected impact? Finally, we observe that alternative worst-case sensor placements may in fact lead to better expected-case performance, such that the 29% increase in expected-case impacts is an upper bound; we explore this issue further in Section 5.

Based on this motivating example, it is natural to ask why the focus should not strictly be on minimization of worst-case performance. In particular, clients have conjectured that minimization of the worst-case impact to "acceptable" levels may require fewer overall sensors than minimization of the expected-case impact, and consequently may be more economically appealing to decision-makers. However, our analyses on Network2 and other test networks support the opposite conclusion. Consider the illustrative situation in which there exist n contamination

events yielding impacts greater than some acceptable threshold T. Further assume that the n events target disparate regions of the network, such that a sensor will mitigate against only one of the n events. In such a situation, n sensors are required to achieve a worst-case impact below T. In contrast, only a small number of sensors $s < n$ may be necessary to yield significant reductions in mean impact, as those sensors are free to be placed at locations in the network capable of detecting contamination from a broad range of events.

Ultimately, there are reasons for studying both expected-case and robust sensor placements. Focusing on expected-case performance is justifiable in situations where a CWS is designed primarily to deal with accidental introduction of contaminants, network hydraulics admit very large numbers of high-impact events, and adversaries are prevented from obtaining knowledge of network structure. In contrast, robust sensor placements should be considered in situations where estimation of contamination event probabilities is difficult, network accessibility is largely unrestricted (e.g., such that injections are easily implemented via backflow), and adversaries can either obtain or infer knowledge of network hydraulics to identify the most damaging injection locations. In reality, CWS designers likely face situations with a combination of these features, such that examination of trade-offs between expected-case and robust sensor placements is necessary.

3 Quantifying Solution Robustness

Informally, "robust" optimization techniques focus on generating solutions that minimize downside risk, i.e., the probability of occurrence of high-consequence events. A common-sense, widely used measure of robustness is that of worst-case cost, which we denote simply as *Worst*. The academic financial community has invested significant effort in developing alternative robust metrics, two of which have gained prominence in the literature: Value-at-Risk (*VaR*) and Tail-Conditional Expectation (*TCE*). Given a set of potential events and their associated costs (e.g., impact to the population in the context of sensor placement), *VaR* is defined as the cost of the $100 \cdot (1 - \alpha)\%$ most costly event (Holton, 2003), where $0 \leq \alpha \leq 1$. Typically, α is taken to be 0.05, such that the minimization of *VaR* effectively allows an optimization algorithm to ignore any costs associated with the $100 \cdot \alpha$ % highest-impact events. *VaR* is an international

standard for risk quantification in the banking community, and has seen widespread application in related contexts. In contrast to *VaR*, *TCE* quantifies the expected cost of the $100 \cdot \alpha\%$ most costly events (Artzner et al., 1999); again, α is typically taken to be 0.05. Consequently, algorithms that minimize *TCE* must make decisions in order to reduce the tail mass of the cost distribution. The conditional value-at-risk measure, denoted *CVaR*, is closely related to the concept of *TCE*. In the case of continuous cost distributions, *CVaR* $=$ *TCE*. In the case of discrete cost distributions, *CVaR* is a continuous approximation to the true cost distribution, such that *TCE* $\leq$ *CVaR*. Overall, we observe that these four risk or robustness measures are related through the following inequality: *VaR* $\leq$ *TCE* $\leq$ *CVaR* $\leq$ *Worst*. The various robust metrics are illustrated graphically in Figure 2.

4 Test Networks and Problem Formulation

We now describe the test networks (Section 4.1), experimental methodology (Section 4.1), and problem formulations (Section 4.2) used to support the motivating analysis presented previously in Section 2 and the more comprehensive analysis presented subsequently in Section 5. The specific algorithms used to solve the sensor placement formulations are described in Section 4.3.

4.1 Networks and Contamination Events

We report computational results for three real, large-scale municipal water distribution networks. The networks are denoted simply as Network1, Network2, and Network3; the identities of the corresponding municipalities are withheld due to security concerns. Network1 consists of roughly 400 junctions, 500 pipes, and a small number of tanks and reservoirs. Network2 consists of roughly 3,000 junctions, 4,000 pipes, and approximately 50 tanks and reservoirs. Network3 consists of roughly 12,000 junctions, 14,000 pipes, and a handful of reservoirs; there are no tanks or well sources in this municipality. All of the models are skeletonized, although the degree of skeletonization in Network1 and Network2 is much greater than in Network3.

Graphical depictions of Network1, Network2, and Network3 are respectively shown in the upper left, upper right, and lower portion of Figure 3. Each graphic was produced by semi-manually "morphing" or altering (e.g., through pipe lengthening or coordinate transla-

tion/rotation) key topological features of the original network structure to further inhibit identification of the source municipalities. Local topologies were largely preserved in this process, such that the graphics faithfully capture the coarse-grained characteristics of the underlying network structures. Sanitized versions of all three networks, in the form of EPANET input files, are freely available from the authors. While these files contain no coordinate information, all data other than that relating to labels (which have been anonymized) are unaltered. Consequently, all computed hydraulic and water quality information accurately reflect (within the fidelity limits of the data and the computational model) the dynamics of the source municipalities. Our goals in making these models available to the broader research community are to facilitate independent replication of our results and to introduce larger, more realistic networks into the currently limited suite of available test problems.

Network hydraulics are simulated over a 96 hour duration, representing four iterations of a typical daily demand cycle. For each junction with non-zero demand, a single contamination event is defined. Each contamination event starts at time $t = 0$ and continues for a duration of 12 hours. Events are modeled as biological mass injections with a constant rate of $5.78e + 10$ organisms per minute. We assume uniform contamination event probabilities, such that all results are normalized by the number of non-zero demand junctions to obtain an expected contamination event impact. Water quality simulations are performed for each event, with a time-step resolution of 5 minutes. The resulting τ_{ej} (as defined in Section 4.2) are then used to compute the impact coefficients d_{ej} for the various design objectives. All hydraulic and water quality simulations are performed using EPANET (Rossman, 2000).

4.2 Optimization Model

To determine an optimal sensor placement x and the corresponding minimal performance metric $f(x)$, we formulate both the expected-case and robust sensor placement problems as Mixed-Integer (Linear) Programs (MIPs), which we then solve using both problem-specific heuristics and a commercially available MIP solver. The MIP-related terms used throughout this paper are defined in the *Mathematical Programming Glossary* (Greenberg, 2006). As we previously showed in (Berry et al., 2006b), the expected-case sensor placement optimization problem is equivalent to the well-known p-median facility location problem. The MIP formulation of the

p-median problem is given as follows, where $\mathcal{E}$ represents the set of contamination events, $\mathcal{L}$ represents the set of network junctions at which a sensor can be placed, p represents the available number of sensors, and q represents a (free) "dummy" sensor that can detect all events given a sufficiently long time horizon (e.g, due to diagnoses at medical facilities):

$$\text{Minimize} \quad \sum_{e \in \mathcal{E}} \sum_{j \in \mathcal{L} \cup \{q\}} d_{ej} x_{ej} \tag{1}$$

$$\text{Subject to} \quad \sum_{j \in \mathcal{L} \cup \{q\}} x_{ej} = 1 \qquad , \forall e \in \mathcal{E} \tag{2}$$

$$x_{ej} \leq y_j \qquad , \forall j \in \mathcal{L}, e \in \mathcal{E} \tag{3}$$

$$\sum_{j \in \mathcal{L}} y_j = p \tag{4}$$

$$y_j \in \{0, 1\} \qquad , \forall j \in \mathcal{L} \tag{5}$$

$$0 \leq x_{ej} \leq 1 \qquad , \forall e \in \mathcal{E}, j \in \mathcal{L} \cup \{q\} \tag{6}$$

The binary y_j variables determine whether a sensor is placed at a junction $j \in \mathcal{L}$. Linearization of the optimization objective is achieved through the introduction of auxiliary variables x_{ej}, which indicate whether a sensor placed at junction j is the first to detect contamination event e. Constraint 3 ensures that detection is possible only if a sensor exists at a junction. The x_{ej} variables are implicitly binary due to a combination of binary y_j, Constraint 3, and the objective function pressure induced by Equation 1. Constraint 4 ensures that exactly p sensors are placed in the network. Constraint 2 guarantees that each contamination event $e \in \mathcal{E}$ is first detected by exactly one sensor, either at q or in the set $\mathcal{L}$; ties are broken arbitrarily. Finally, the objective function (Equation 1) ensures that detection of an event e is assigned to the junction $j \in \mathcal{L} \cup \{q\}$ such that d_{ej} is minimal.

The impact of a potential contamination event is determined via transport simulation. EPANET (Rossman, 2000) is used to generate a time-series τ_{ej} of contaminant concentration at each junction $j \in \mathcal{L}$ for each event $e \in \mathcal{E}$. The resulting time-series are then used to compute the network-wide impact d_{ej} of the event e assuming first detection via a sensor placed at junction j. More formally, let γ_{ej} denote the earliest time t at which a sensor at junction j can detect

contamination due to event e, e.g., when contaminant concentration reaches a specific detection threshold. If contaminant from event e fails to reach junction j, then $\gamma_{ej} = t^*$, where t^* denotes either the end of the simulation; otherwise, γ_{ej} can easily be computed from τ_{ej}. Further, let $d_e(t)$ denote the network-wide damage incurred by an event e up to time t. Next, we define $d_{ej} = d_e(\gamma_{ej})$, i.e., the aggregate, network-wide damage incurred if event e is first detected at time γ_{ej}. In our analysis, $d_{sq} = d_s(t^*)$. We assume without loss of generality that a sensor placed at a junction $j \in \mathcal{L}$ is capable of immediately detecting any contamination from event $e \in \mathcal{E}$ – assuming the contaminant can reach junction j – once non-zero concentration levels of a contaminant are present. In the absence of realistic alarm procedures and mitigation strategies, we assume that both consumption and propagation of contaminant is terminated once detection occurs; extensions to deal with delayed notification are described in (Berry et al., 2006b). Finally, we observe that the p-median optimization formulation – through the use of d_{ej} coefficients – allows for the use of arbitrarily complex contamination events, e.g., multiple simultaneous injection sites with different contaminants at variable injection strengths and durations.

We have also investigated extensions of the basic MIP formulation to robust metrics. While expression of a MIP formulation to minimize *Worst* is a straightforward extension of the expected-case formulation, the *CVaR* (the continuous approximation to *TCE*, which in general is discretized) formulation is significantly more complicated. For reasons discussed in below in Section 4.3, we do not discuss these formulations herein, and instead refer to Greenberg et al. (2007).

We quantify the impact due to a contamination event as the number of individuals sickened by exposure prior to detection by either a sensor or a sufficient time delay (i.e., detection by the dummy sensor q). The specific computation is defined via the demand-based model (in which contaminant ingestion is proportional to volume of water extracted from a distribution system) described in Murray et al. (2006), and the values for the numerous parameters in the dosage-response computation can be obtained from the authors. The Murray et al. (2006) model yields potentially fractional population counts, but to simplify the presentation we round all reported values to the nearest integral value. Alternative models of population exposure have assumed the availability of population estimates on a time-varying, per-junction basis (Berry et al., 2005a;

Watson et al., 2004). While correcting the obvious deficiency of demand-based models, reliable estimates of time-varying population density are generally unavailable.

4.3 Algorithms

We have previously described both heuristic and exact algorithms for solving expected-case MIP formulations of the sensor placement problem (Berry et al., 2006b). We employed commercially available, state-of-the-art MIP solvers, specifically ILOG's CPLEX 10.0 solver[1], to compute provably optimal solutions. Using various modeling techniques to reduce the size of the basic formulation, we were able to identify optimal solutions to Network3 (our largest test network) in roughly 15 minutes of CPU time on a modern computing workstation. These techniques take advantage of equality in the arrival time of contaminant at various junctions, due to the imposition of a discretized water quality time-step. Consequently, the impacts d_{ej} are identical for various junctions j, which can be collected into "superlocations", thereby reducing the effective size of the formulation (Berry et al., 2007).

We also applied a Greedy Randomized Adaptive Search Procedure (GRASP) to heuristically generate high-quality solutions to the expected-case MIP formulation. The algorithm, fully described in Resende and Werneck (2004), is a simple multi-start local search procedure in which steepest-descent hill-climbing is applied to a number N of initial solutions. The local search neighborhood used in the GRASP algorithm is based on sensor exchange: each "move" consists of removing a sensor from a junction and placing it at a junction without a sensor. The steepest-descent procedure selects the exchange that results in the largest increase in performance at each iteration, and terminates once no improvements are possible. The best of the N solutions is returned by the algorithm. Our experiments indicate that the GRASP heuristic obtains solutions significantly faster than the MIP solves described above, e.g., in under three minutes for Network3. Further, in all cases investigated to date, the obtained solutions were optimal, i.e., equivalent in quality to those obtained by CPLEX.

We extended the GRASP heuristic to enable solution of the robust variants of the MIP formulation described in Section 4.2. The extensions involved modification of the move evaluation code that determines the change in performance associated with simultaneously removing

[1] http://www.ilog.com

a sensor from junction x and placing it instead at an open junction y. The efficiency of the resulting heuristic is dictated by the speed of move evaluation, which can be accelerated by various analytic techniques specific to the p-center and related facility location problems; we defer to Mladenovic et al. (2003) for a discussion of these techniques.

5 Expectation versus Robust Sensor Placements

We now examine the performance differences between expected-case and robust sensor placements on our test networks. Our analysis is broken into two components. We begin in Section 5.1 by expanding the motivational analysis presented in Section 2 to additional robustness measures and test networks. In Section 5.2, we then discuss several key qualitative differences between expected-case and robust placements in terms of sensor locations in Network2.

5.1 A Quantitative Analysis of Placement Characteristics

For each of our test networks, we use the heuristic algorithm described in Section 4.3 to develop sensor placements that attempt to independently minimize *Mean* performance and the various robust metrics. As discussed in Section 6, we cannot in general guarantee the optimality of robust sensor placements due to the increased difficulty of the corresponding robust MIP formulations relative to the baseline expected-case MIP formulation. The performance of each of the resulting sensor placements is then quantified in terms of the *Mean*, *VaR*, *TCE*, and *Worst* metrics. The results for Network1 through Network3 are respectively shown in Tables 1 through 3. We observe that in each of the tables, the inequality $\mathit{VaR} \leq \mathit{TCE} \leq \mathit{Worst}$ holds, as required, for the diagonal entries.

We first consider the results for Network1 (see Table 1), in which 5 sensors are placed to protect against 105 contamination events; contamination events are initiated at each of the 105 out of approximately 400 junctions with non-zero demand. Due to the small scale of this problem, we are able to establish the optimality of the *Worst* sensor placement by exactly solving the MIP formulation; we were unable to establish optimality for the *TCE* sensor placement. Relative to the example shown in Section 2, we observe even more dramatic differences between the *Mean* and *Worst* sensor placements: the worst-case impact can be cut in half for less than a 13%

increase in the mean impact. Via exhaustive enumeration of the solution space via a modified MIP branch-and-bound procedure, we determined that there are in fact a number of *alternative global optima* that satisfy *Worst* $= 605$. This finding raises the possibility that solutions with *Worst* $= 605$ and *Mean* ≤ 162 (13% above the minimal 143 value) may exist. Indeed, using a modified version of our heuristic algorithm that allows for specification of side constraints, we found such a solution with *Worst* $= 605$ and *Mean* $= 148$; the latter represents roughly a 3% increase relative to the optimal value of *Mean* $= 143$. This observation further illustrates the degree to which it is possible to trade off robust versus expected-case performance; in particular, it seems likely that decision-makers would prefer this particular *Worst* placement over the optimal *Mean* placement.

Although we could in principle perform a similar analysis for each of the results shown in Tables 1 through 3, side constraints further increase the difficulty of the robust MIP formulations, which as discussed in Section 6 is already substantial. Rather, we simply note that optimality (or presumed optimality) with respect to one metric does *not* guarantee conditional optimality (e.g., optimal on a secondary measure given a constraint on a primary measure) on the complementary measures, due to the potential presence of alternative optima. Finally, we observe that although the performance characteristics of the *Mean* and *Worst* placements are significantly different, the placements themselves are not; the two *Worst* placements discussed above locate sensors at respectively two and three of the junctions at which sensors are located in the *Mean* placement.

Given that *VaR*, *TCE*, and *Worst* all quantify related aspects of the distribution of strictly high-impact contamination events, we expected *a priori* that sensor placements minimizing these robustness measures would be strongly correlated in terms of their performance, i.e., sensor placements yielding minimal performance with respect to one robust metric will yield near-minimal performance in terms of all robust metrics. Unexpectedly, the data shown in Table 1 indicate this is not the case. For example, the *Worst* performance of the *VaR*-optimal placement is more than double that of the optimal *Worst* performance. Even discounting potential effects due to alternative global optima, the effect remains significant; minimizing *Worst* subject to *VaR* ≤ 388 yields only a slight reduction in *Worst*, to 1249. Similar discrepancies exist between the observed and optimal values of *TCE* given a *VaR*-optimal placement. Of course, minimiza-

tion of *VaR* allows for any distribution of the remaining α proportion of high-impact events, so the results are consistent. However, the degree of the divergence was unexpected. In general, this behavior simply reinforces the importance of understanding and analyzing the performance metrics used in optimization; apparently subtle definitional differences (e.g., between *TCE* and *Worst*) in metrics can yield significant differences in both sensor placements and performance.

Next, we consider the results for Network2 (see Table 2), which extends the analysis presented in Section 2 to other robust metrics; we are unable to establish optimality of any of the robust sensor placements for Network2 and Network3. Expanding on the previously noted observation that trade-offs in *Mean* and *Worst* performance are possible, we again observe alternative optima in this problem for the *Worst*-optimal performance. Mirroring the approach discussed above for Network1, we were able to generate a solution via imposition of side constraints with *Worst* $= 3490$ and *Mean* $= 768$, in contrast to the initial value of *Mean* $= 869$ given the *Worst*-optimal solution. Consequently, it is possible in Network2 to obtain a nearly 30% reduction in worst-case impact at the expense of a relatively minor 12% increase in mean impact. Interestingly, despite similar performance, this solution and the *Mean*-optimal solution share sensors at only *two* of the possible twenty junctions in common. Finally, as with the results for Network1, the performance of the robust metrics is not strongly correlated – even accounting for the presence of alternative global optima.

We conclude by noting that results analogous to those observed for Network1 and Network2 extend to Network3, the results for which are shown in Table 3. Overall, our primary conclusions – (1) that it is possible to trade off expected-case and robust performance and (2) the performance of various robust sensor placements is not strongly correlated – hold over a range of distribution network scales, from very small municipalities to large-scale cities. Consequently, the issues we raise in our analysis are broadly applicable to decision-makers in the water security domain.

5.2 A Qualitative Assessment of Placement Characteristics

Quantitative analysis is only one avenue to understanding and exploring the relationships between expected-case and robust sensor placements. In this section, we compare and contrast the qualitative characteristics of expected-case and worst-case sensor placements for Network2,

each containing 20 sensors. The locations of the corresponding sensor placements are respectively shown in the left and right sides of Figure 4. In Network2, water is treated at a single source (located in the lower right sides of the graphics shown in Figure 4) and pumped in stages to successively higher elevations. To compare the two sensor placements, we consider characteristics such as the size and number of pipes connected to the sensor junctions, in addition to the demand at sensor junctions. Further, we consider the number of contamination events that are detected by each sensor placement, the average impact of these contamination events, and the time to detection.

In both placements, the sensors are located at junctions along relatively large diameter pipes, which are additionally often connected to more than two pipes; about half of the sensors are located at junctions with large demand. Specifically, all sensors are located on junctions connected to 8 inch or larger diameter pipes, which is the median diameter of pipes in Network2. Moreover, the majority of sensor-equipped junctions are connected to 12 inch pipes or greater (17-18 of the 20). One difference in the two placements, however, is that the *Worst* placement locates half of the sensors on junctions connected to 20 inch or larger pipes, while only 25% of the sensors in the *Mean* placement are connected to 20 inch pipes or larger.

In both placements, 17 of the 20 sensors are located on junctions connected to 3 or more pipes; none are placed at dead-end junctions. Further, 8 of the 20 sensors in both placements are located on junctions in the top quartile of demands. For the expected-case placement, 5 sensors are located at zero-demand junctions, while for the *Worst* placement, 8 sensors are located at zero-demand junctions.

It appears from examination of Figure 4 that sensors in the *Worst* placement are somewhat closer together, possibly resulting in less spatial coverage of the distribution network. However, approximately the same number of contamination events are detected by physical sensors: roughly 850 events out of a total of approximately 1,600. Each sensor is responsible for detecting approximately 42 contamination events on average. The average time to detection for each placement is similar; 7 hours for the *Mean* placement and 9 hours for the *Worst* placement. However, the average impact of the contamination events at the time of detection by the *Mean* placement is 685 individuals, in contrast to 882 individuals for the *Worst* placement. It is also interesting to note that a sensor is located much closer to the water source in the *Worst*

placement, but not in the *Mean* placement.

In summary, the two sensor placements are surprisingly quite similar in terms of the diameter of connected pipes, the number of connected pipes, the demand at the sensor junctions, and the number of contamination events detected. Notable differences are that the *Worst* solution places more sensors on larger-diameter pipes, does not demonstrate an even spatial spread throughout the network, and has a sensor located much closer to the source. Further, the *Worst* placement allows for significantly higher impacts on average (which is obviously necessary to achieve low *Worst* performance), but counterintuitively takes longer on average to detect contamination events. Overall, subtle differences in sensor locations appear to be responsible for the large observed discrepancies in terms of both *Mean* and *Worst* performance.

6 Computational Experience

We now analyze the computational properties of the GRASP heuristic and the MIP models described in Section 4.3, contrasting differences between expected-case and robust optimization models. As hinted at previously, robust MIP formulations are empirically much more difficult to solve than their expected-case counterparts. To quantify this discrepancy, we consider the average run-times required to generate a single local optimum using the GRASP heuristic for the *Mean*, *VaR*, *TCE*, and *Worst* performance metrics. Our computational platform is a workstation containing a 64-bit AMD 2.2GHz Opteron CPU running the Linux 2.6 operating system; the platform possess 64GB of RAM, such that run-time issues relating to memory paging are non-existent. All codes were written in C++ and compiled using the GNU gcc compiler. The results for all three of our test networks are shown in Table 4, using the sensor budgets indicated in Section 4.1. The run-times include the time required to load the problem instance.

The results clearly illustrate the difficulty of robust variants of the sensor placement problem. Although Network1 run-times are clearly negligible for any metric, the divergence between the *Mean* and other metrics is significant for Network2; the run-times under the *Mean* and *Worst* metrics differ by a factor of 100, and are even larger under the *VaR* and *TCE* metrics. Relative to Network1, the growth in difficulty is accentuated in part due to the growth in the sensor budget p, as the number of exchanges available from any solution is a monotonically

increasing function of both $|\mathcal{L}|$ and p for the range of p we consider. Even larger, analogous discrepancies are observed for Network3, where the run-times under the *Mean* and *Worst* metrics differ by a factor of nearly 2,500. The difficulty of computing samples for the *VaR* and *TCE* metrics is much greater than that for *Worst*. This is due to the additional need for sorting the impacts (in the case of *VaR* and *TCE*) and computing the tail expectation (in the case of *TCE*).

We now consider the relative difficulty of expected-case and robust MIP formulations for exact solvers. Specifically, we executed CPLEX 10.0 on each of our test networks, to independently minimize *Mean*, *CVaR*, and *Worst*. The computational platform was identical to that described above for the heuristic tests, and a limit of 24 (and in some cases greater, for Network3) hours was imposed on each individual run. The results are reported in Table 5.

We first examine the results for Network1, observing that minimization of the robust metrics requires several orders of magnitude more run-time than required for the *Mean* metric. However, minimization of *CVaR* is less costly than *Worst*. We currently have no explanation for this discrepancy. Next, we examine the results for Network2 and Network3. In no case could CPLEX minimize the robust metrics within the allocated time limit. In several cases, a feasible solution could not be located, and in all cases the heuristic solutions yielded better performance than the best solution found by CPLEX. Overall, these results clearly reinforce the dramatic differences in difficulty involved in minimization of expected-case versus robust performance metrics; the latter require at least 20 times more computational effort, and in most cases, significantly more.

Overall, the data presented in Tables 4 and 5 illustrate the challenges associated with optimization of robust performance metrics. Although MIP methods are tractable in the case of minimizing *Mean* impacts, optimal robust solutions - or at least proofs of optimality - are currently out of reach of exact methods. Even with heuristics, locating high-quality solutions to robust formulations requires a significant computational investment. However, even lacking optimal solutions, the fundamental conclusions presented in Section 5 still hold: it is possible to trade off expected versus robust performance. Future improvements in heuristic and exact technologies will further enhance our ability to exploit this property, and to better understand the relationship between the various robust metrics. Finally, we observe that the relative difficulty of robust optimization is not necessarily inherent. Our results are empirical, rather than

theoretical, and it is possible that additional research will expose techniques for significantly improving algorithm performance, e.g., cuts in the case of MIPs or more effective move evaluators in the case of heuristics. Algorithms for minimizing the expected case, i.e., for solving the p-median formulation, have been extensively studied for decades, and only recently have these algorithms yielded results as impressive as those we report.

7 Conclusions

Most extant techniques for the sensor placement problem in water distribution networks consider minimization of the expected impact of a contamination event. However, the solutions generated by these techniques admit a number of low-probability, very high-impact contamination events. The presence of these events, in addition to consideration of known inaccuracies in and difficulties associated with contamination event probability estimation, should motivate decision makers to assess the differences between solutions that minimize expected impact and those that focus strictly on high-consequence contamination events.

We introduce a number of so-called robust metrics for quantifying the impact of high-consequence contamination events. Using both heuristic and exact optimization techniques, we then contrast the performance characteristics of solutions that respectively attempt to minimize the mean and robust metrics. We show that it is possible to gain significant reductions in the number and degree of high-consequence events, at the expense of moderate increases in the mean impact of a contamination event. Additionally, we find that performance with respect to different robust metrics is not highly correlated, further emphasizing the need to develop a deeper understanding of the relationship between solutions developed using different robust metrics.

Finally, we demonstrate that solution of robust sensor placement problems is significantly more difficult than solution of expected-case sensor placement problems. Although heuristics can identify high-quality solutions for robust formulations, exact methods are only able to tackle the smallest test networks. New research effort will ultimately be required to develop truly efficient techniques for solving robust problems.

Acknowledgments

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. This work was funded through an Interagency Agreement with the United States Environmental Protection Agency's National Homeland Security Research Center. The authors acknowledge assistance from Harvey Greenberg and Tod Morrison (for model development), in addition to Lee Ann Riesin and Jonathan Berry (for software and algorithmic support).

References

Artzner, P., Delbaen, F., Eber, J., and Heath, D. (1999). Coherent measures of risk. *Mathematical Finance*, 9:203–228.

Berry, J., Carr, R., Hart, W., Leung, V., Phillips, C., and Watson, J. (2006a). On the placement of imperfect sensors in municipal water networks. In *Proceedings of the 8th Symposium on Water Distribution Systems Analysis*.

Berry, J., Carr, R., Hart, W., and Phillips, C. (2007). Scalable water network sensor placement via aggregation. In *Proceedings of the ASCE/EWRI Congress*.

Berry, J., Fleischer, L., Hart, W., Phillips, C., and Watson, J. (2005a). Sensor placement in municipal water networks. *Journal of Water Resources Planning and Management*, 131(3):237–243.

Berry, J., Hart, W., Phillips, C., Uber, J., and Walski, T. (2005b). Water quality sensor placement in water networks with budget constraints. In *Proceedings of the ASCE/EWRI Congress*.

Berry, J., Hart, W., Phillips, C., Uber, J., and Watson, J. (2006b). Sensor placement in municipal water networks with temporal integer programming models. *Journal of Water Resources Planning and Management: Special Issue on Drinking Water Distribution Systems Security*, 132:218–224.

Carr, R., Greenberg, H., Hart, W., Konjevod, G., Lauer, E., Lin, H., Morrison, T., and Phillips,

C. (2006). Robust optimization of contaminant sensor placement for community water systems. *Mathematical Programming*, 107(1):337–356.

Greenberg, H. (1996–2006). *Mathematical Programming Glossary*. World Wide Web, `http://www.cudenver.edu/~hgreenbe/glossary/`.

Greenberg, H., Morrisson, T., and Hart, W. (2007). Robust MIP formulations for sensor placement optimization. Technical report, Sandia National Laboratories.

Holton, G. A. (2003). *Value-at-Risk: Theory and Practice*. Academic Press.

Kessler, A., Ostfeld, A., and Sinai, G. (1998). Detecting accidental contaminations in municipal water networks. *Journal of Water Resources Planning and Management*, 124(4):192–198.

Mladenovic, N., Labbe, M., and Hansen, P. (2003). Solving the p-center problem with tabu search and variable neighborhood search. *Networks*, 42(1):48–64.

Murray, R., Uber, J., and Janke, R. (2006). Modeling acute health impacts resulting from ingestion of contaminated drinking water. *Journal of Water Resources Planning and Management: Special Issue on Drinking Water Distribution Systems Security*, 132:293–300.

Ostfeld, A. and Salomons, E. (2004). Optimal layout of early warning detection stations for water distribution systems security. *Journal of Water Resources Planning and Management*, 130(5):377–385.

Resende, M. and Werneck, R. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88.

Rossman, L. (2000). The EPANET programmer's toolkit for analysis of water distribution systems. In *Proc. of the Annual Water Resources Planning and Management Conference*.

USEPA (2005). WaterSentinel System Architecture. Technical report, U.S. Environmental Protection Agency.

Watson, J., Greenberg, H., and Hart, W. (2004). A multiple-objective analysis of sensor placement optimization in water networks. In *Proceedings of the ASCE/EWRI Congress*.

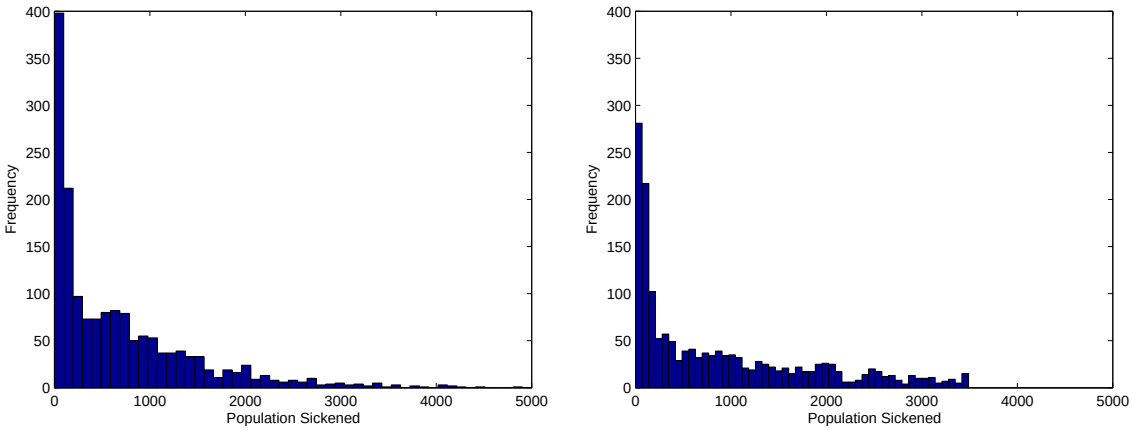

Figure 1: Histograms of the number of individuals sickened for various contamination events in Network2 under expected-case (left figure) and worst-case (right figure) sensor placements.

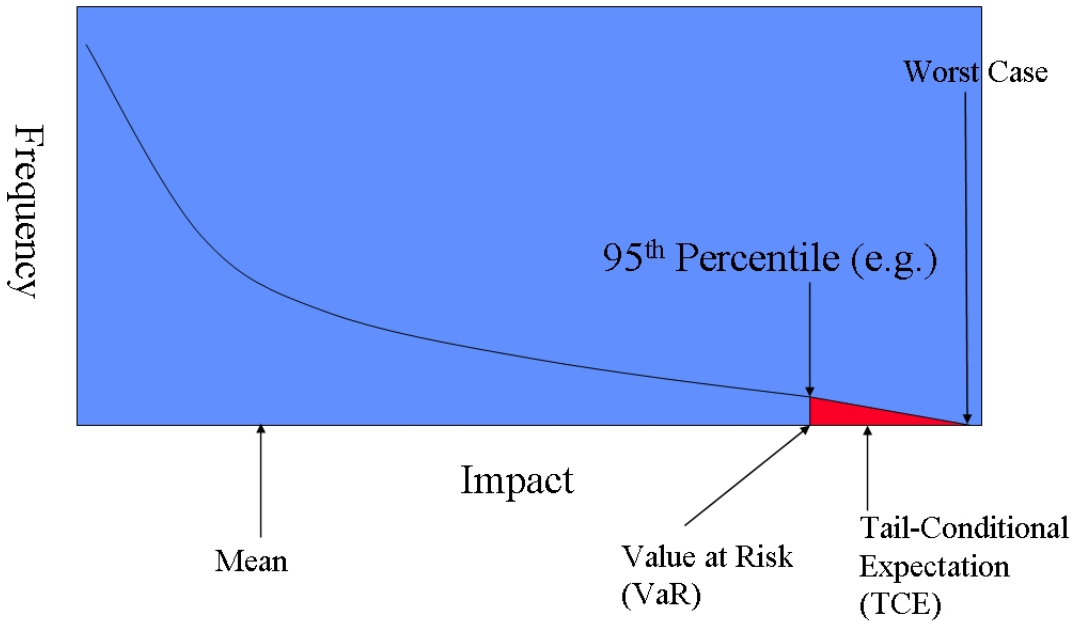

Figure 2: Graphical illustration contrasting the various "robust" metrics of sensor placement performance.

	Performance Metric			
Objective to Minimize	*Mean*	*VaR*	*TCE*	*Worst*
Mean	**143**	476	749	1249
VaR	175	**388**	824	1447
TCE	190	476	**539**	679
Worst	162	565	587	**605**

Table 1: Performance of expected-case and robust sensor placements in terms of various metrics for Network1, generated using the GRASP heuristic. The placements consist of 5 sensors mitigating against 105 possible contamination events.

Figure 3: Graphical depictions of Network1 (upper left), Network2 (upper right), and Network3 (lower) municipality distribution topologies.

	Performance Metric			
Objective to Minimize	*Mean*	*VaR*	*TCE*	*Worst*
Mean	**685**	2244	2953	4902
VaR	740	**2019**	2699	5076
TCE	757	2112	**2508**	3962
Worst	869	2773	2990	**3490**

Table 2: Performance of expected-case and robust sensor placements in terms of various metrics for Network2, generated using the GRASP heuristic. The placements consist of 20 sensors mitigating against 1621 possible contamination events.

	Performance Metric			
Objective to Minimize	*Mean*	*VaR*	*TCE*	*Worst*
Mean	**320**	1214	1767	4780
VaR	335	**1188**	1781	5794
TCE	343	1283	**1685**	4219
Worst	463	1934	2315	**3079**

Table 3: Performance of expected-case and robust sensor placements in terms of various metrics for Network3, generated using the GRASP heuristic. The placements consist of 20 sensors mitigating against 9705 possible contamination events.

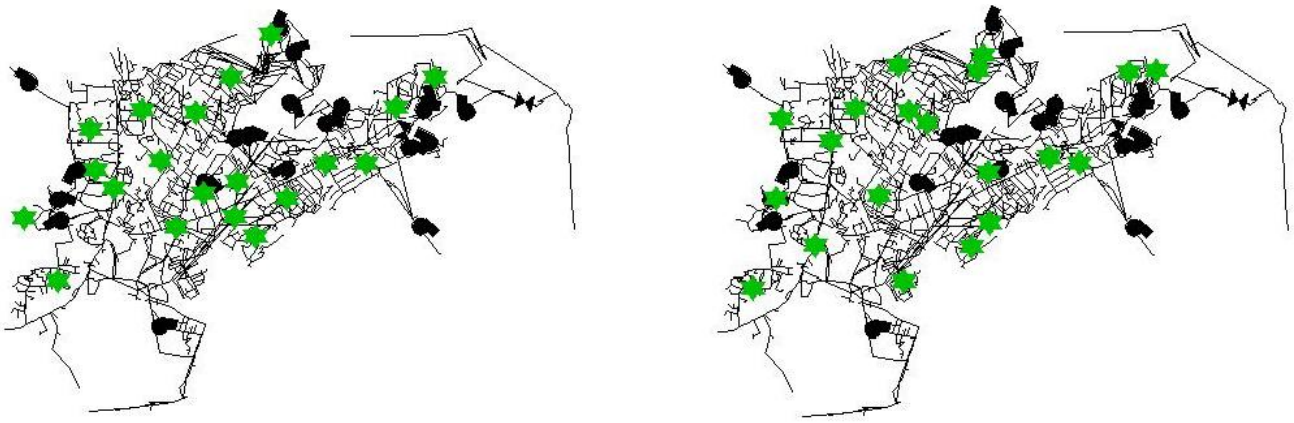

Figure 4: The location of sensors corresponding to *Mean* (left figure) and *Worst* (right figure) sensor placements for Network2. Junctions with sensors are denoted by "star"-shaped graphical overlays.

	Mean Run-Time per Local Optimum		
Objective to Minimize	Network1	Network2	Network3
Mean	0.01s	0.81s	6.5s
Worst	0.02s	97s	4.4hrs
VaR	0.05s	643s	20.4hrs
TCE	0.06s	810s	26.0hrs

Table 4: Mean run-times required for the GRASP heuristic to generate a local optimum to both expected-case and robust variants of the sensor placement problem, for each of our test networks.

	Run-Time		
Objective to Minimize	Network1	Network2	Network3
Mean	0.70s	3m2s	47m31s
Worst	8m20s	>24hrs	>48hrs
CVaR	3m18s	>24hrs	>96hrs

Table 5: Run-times to solve the exact MIP models for expected-case and robust variants of the sensor placement problem, for each of our test networks.

B Pyomo Technical Report

Python Optimization Modeling Objects (Pyomo)

Nicolas L. Benavides* Robert D. Carr† William E. Hart‡

September 14, 2007

Abstract

We describe the Python Optimization Modeling Objects (Pyomo) package. Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Pyomo provides a capability that is commonly associated with algebraic modeling languages like AMPL and GAMS. We introduce Pyomo by contrasting it with the capabilities of AMPL.

1 Introduction

Algebraic Modeling Languages (AMLs) are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [9]. AMLs like AIMMS [1], AMPL [2, 8] and GAMS [5] have programming languages with an intuitive mathematical syntax that supports concepts like sparse sets, indices, and algebraic expressions. AMLs provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is almost essential for real-world problems that can involve thousands of constraints and variables.

An alternative strategy for modeling mathematical problems is to use a standard programming language in conjunction with a software library that uses object-oriented design to support similar mathematical concepts. Although these modeling libraries sacrifice the intuitive mathematical syntax of an AML, they allow the user to leverage the greater flexibility of standard programming languages. For example, modeling libraries like FLOPC++ [4], OPL [6] enable the solution of large, complex problems within a user-defined application.

This paper describes Pyomo, the Python Optimization Modeling Objects (Pyomo) package. Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Like other modeling libraries, Pyomo can generate problem instances and apply optimization solvers with a fully expressive programming language. Further, Python is a noncommercial language with a very large user community, which will ensure robust support for this language on a wide range of compute platforms.

Python is a powerful dynamic programming language that has a very clear, readable syntax and intuitive object orientation. Python's clean syntax allows Pyomo to express mathematical concepts with a reasonably intuitive syntax. Further, Pyomo can be used within an interactive Python shell, thereby allowing a user to interactively interrogate Pyomo-based models. Thus, Pyomo has many of the advantages of both AML interfaces and modeling libraries.

Pyomo makes a clear distinction between the abstract specification of a model, generation of model instances, and the solution of model instances. Abstract models are a key element of AML's like AMPL, and this capability clearly distinguishes Pyomo from other Python modeling libraries like CVXOpt [3] and PuLP [7]. Pyomo models can be solved with either Python optimizers, or with externally defined solvers

*Santa Clara University, NBenavides@scu.edu
†Sandia National Laboratories, rdcarr@sandia.gov
‡Sandia National Laboratories, wehart@sandia.gov

(e.g. GLPK, CPLEX and CBC). Further, Python can integrate extension modules in low level languages like C or C++ to directly leverage fast solver libraries, and wrapped modules can be used within Python exactly like native Python code.

Section 2 illustrates how Pyomo would be used to model a simple application. We compare and contrast the Pyomo formulation with a formulation developed in the widely used AMPL modeling language. Section 3 describes the Pyomo classes that are used to define model components.

2 A Simple Example

In this section we illustrate Pyomo's syntax and capabilities by demonstrating how a simple AMPL example can be replicated with Pyomo Python code.

Consider the basic AMPL program prod.mod:

```
set P;

param a {j in P};
param b;
param c {j in P};
param u {j in P};

var X {j in P};

maximize Total_Profit: sum {j in P} c[j] * X[j];

subject to Time: sum {j in P} (1/a[j]) * X[j] <= b;

subject to Limit {j in P}: 0 <= X[j] <= u[j];
```

To translate this into Pyomo, the user must first import the Pyomo module and create a Pyomo **Model** object:

```
#
# Import Pyomo
#
from pyomo import *

#
# Create model
#
model = Model()
```

This import assumes that Pyomo is available on the users's Python path (see Python documentation for PYTHONPATH for further details). Next, we create the sets and parameters that correspond to the data used in the AMPL model. This can be done very intuitively using the **Set** and **Param** classes.

```
model.P = Set()

model.a = Param(index=model.P)
model.b = Param()
model.c = Param(index=model.P)
model.u = Param(index=model.P)
```

Note that parameter b is a scalar, while parameters a, c and u are arrays indexed by the set P. Pyomo also defines the **ProductSet** class, which can be defined in a similar manner.

Next, we define the decision variables in this model.

```
def X_bounds(j, model):
    return (0,model.u[j])
model.X = Var(index=model.P, bounds=X_bounds)
```

Decision variables and model parameters are used to define the objectives and constraints in the model. Parameters define constants and the variables are the values that are optimized. Parameter values are typically defined by a data file that is processed by Pyomo.

Objectives and constraints are explicitly defined expressions in Pyomo. The **Objective** and **Constraint** classes require a **rule** option that specifies how these expressions are constructed. This is a function that takes one or more arguments: the first arguments are indices into a set that defines the set of objectives or constraints that are being defined, and the last argument is the model that is used to define the expression.

```
def Objective_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + model.c[j] * model.X[j]
    return ans
model.profit = Objective(rule=Objective_rule)

def Time_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + (1.0/model.a[j]) * model.X[j]
    return ans < model.b
model.Time = Constraint(rule=Time_rule)
```

The rules used to construct these objects use standard Python functions. Finally, note that the **Time_rule** function includes the use of $<$ and $>$ operators on the expression. These operators are used to define upper and lower bounds on the constraints.

Once an abstract model has been created, it can be printed as follows:

```
print ''ABSTRACT MODEL''
model.pprint()
```

This summarize the information in the Pyomo model, but it does not print out explicit expressions. This is due to the fact that an abstract model needs to be instanted with data to generate the model objectives and constraints:

```
instance = model.create(''prod.dat'')

print ''MODEL INSTANCE''
instance.pprint()
```

Appendix A shows the final Python code for this example.

Once a model instance has been constructed, an optimizer can be applied to it to find an optimal solution. For example, the PICO integer programming solver can be used within Pyomo as follows:

```
opt = solvers.PICO(path="/home/wehart/bin/PICO",keepFiles=True)
solutions = opt.solve(instance)
```

This creates an optimizer object for the PICO executable defined in a given path, and it indicates that temporary files should be kept. The Pyomo model is handed to this optimizer, which returns the final solutions generated by the optimizer.

3 Documentation of Pyomo Objects

In this section we provide more detail on the definitions of Pyomo classes that are used to define models.

3.1 Sets

The **Set()** class is used to index other objects (e.g. **Param** and **Var**). This class has the same look-and-feel as a **sets.Set** class, but it can be used to define an abstract set. This class contains a concrete set, which can be initialized by the **load()** method, or directly.

Constructor arguments:

- within - A set that defines the type of values that can be contained in this set
- default - Default set members, which may be overriden when setting up this set
- rule - A rule for setting up this set with existing model data. This has the functional form: f: pyomo.Model $->$ pyomo.Set
- restriction - Define a rule for restricting membership in a set. This has the functional form: f: data $->$ bool and returns true if the data belongs in the set

3.2 Product Sets

The **ProductSet()** class represents the cross product of other sets.

Constructor arguments:

- default - Default set members, which may be overriden when setting up this set
- rule - A rule for setting up this set with existing model data. This has the functional form: f: pyomo.Model $->$ pyomo.Set
- restriction - Define a rule for restricting membership in a set. This has the functional form: f: data $->$ bool and returns true if the data belongs in the set

In the following AMPL code, the **rate** paramater's index set is the cross product of two sets:

```
set PROD;
set STAGE;

param rate {PROD,STAGE};
```

In Pyomo, the cross product is created with the **ProductSet** class, and the result of this is used to index other Pyomo objects:

```
model.PROD = Set()
model.STAGE = Set()

model.setprod = ProductSet( (model.PROD, model.STAGE) )
model.rate = Param(index=model.setprod)
steel4mod.rate > 0
```

3.3 Parameters

The **Param()** class defines constant values in a model, and a parameter object may be defined over an index.

Constructor arguments :

- index - The index set that defines the distinct parameters. By default, this is None, indicating that there is a single parameter.
- domain - A set that defines the type of values that each parameter must be.
- validate - A rule for validating this parameter with respect to data that exists in the model
- default - A set that defines default values for this parameter
- rule - A rule for setting up this parameter with existing model data

3.4 Variables

The **Var()** class defines a numeric variable, which may be defined over an index.

Constructor arguments:

- index - The index set that defines the distinct variables. By default, this is None, indicating that there is a single variable.
- domain - A set that defines the type of values that each parameter must be.
- default - A set that defines default values for this variable
- bounds - A function that defines bound constraints for this variable

Simple bound constraints on variables can be specified with the **bounds** rule:

```
model.P     = Set()

model.x_lb = Param(index=model.P)
model.x_ub = Param(index=model.P)

def x_bounds(i, model):
    return (model.x_lb[i], model.x_ub[i])
model.x = Var(index=model.P, rule=x_bounds)
```

3.5 Objectives

The **Objective()** class defines an objective expression.

Constructor arguments:

- rule - A rule for constructing this objective with existing model data.
- sense - Used to define wether this objective should be minimized or maximized (minimization is the default).

3.6 Constraints

The **Constraint()** class defines an expression whose value is constrained in the model.

Constructor arguments:

- rule - A rule for constructing this constraint with existing model data.
- index - Defines a set of constraints over an index.

Note that the **rule** option generally needs to include a definition of the bounds on a constraint. A constraint must have either an upper or lower bound, and it may have both. For example:

```
model.P = Set()
model.Q = Set()

model.x = Var(index=model.Q)

def c_rule(i, model):
    ans = 0
    for q in model.Q:
      ans = ans + model.x[q]
    ans = ans > 0
    return ans < 1
model.c = Constraint(index=model.P, rule=c_rule)
```

The last two lines in the **c_rule** function define upper and lower bound values for the **c** constraint. Note that this is a non-standard use of the $<$ and $>$ operators; these operators return an expression rather than a boolean value.

3.7 Models

The **Model()** class defines a mixed-integer model that can be optimized by a user. This class takes no arguments, but it is a container for instances of the other Pyomo objects created by the user. For example, consider the statement:

```
model.x = Var()
```

This statement registers the variable x in the model, and assigns it the name "x".

4 Conclusions

Pyomo has many of the features of abstract modeling languages and optimization modeling libraries, but the following features of Pyomo are noteworthy:

- Pyomo supports the ability to define abstract problems from which problem instances can be generated. Further, Pyomo can generate multiple instances, which can be analyzed simultaneously in separate Python class objects.
- Pyomo is based on a powerful, commonly available open-source language. Thus, there are no licensing limitations with the use of Pyomo, and the set of Pyomo objects can be customized for an application in ways that are not possible with commercial AMLs and modeling libraries.
- Python has a clean syntax, so Pyomo modeling objects can be used in an intuitive manner.

- Pyomo models can leverage Python's programming language to define complex data structures and standard programming constructs like classes and functions. Further, Python can be naturally linked with external libraries for high-performance kernels.
- Pyomo can integrate optimization solvers in an extensible manner. Optimizers can be defined within Python itself, and external optimizers can be launched using file I/O to communicate with Python.[1]

Pyomo is probably most similar to the FLOPC++ modeling library. FLOPC++ is writen in C++, and it has many of the same objects as are used in Pyomo. While FLOPC++ enables models to be embedded in compiled application codes, Pyomo enables the rapid prototyping of models in a scripting language. Thus, these capabilities seem quite complementary.

The current implementation of Pyomo has been validated on a small set of simple models. In the future, more extensive validation of Pyomo is needed to ensure that it can express a wide range of complex problems. Further, the performance of Pyomo needs to be analyzed to ensure that it can effectively generate large-scale optimization models. Finally, this document needs to be extended to include examples that illustrate how Pyomo can leverage Python to develop complex models more naturally than AMLs like AMPL and GAMS.

This document describes the initial prototype of Pyomo. Once this code has stablized, we plan to integrate Pyomo into the COIN-OR optimization software repository to encourage its use within the academic and business communities.

Acknowledgements

Thanks to Jon Berry and Cindy Phillips for their critical feedback on the design of Pyomo. 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.

References

[1] *AIMMS home page.*

[2] *AMPL home page.*

[3] *CVXOPT home page.*

[4] *FLOPC++ home page.*

[5] *GAMS home page.*

[6] *OPL home page.*

[7] *Pulp: A python linear programming modeler.*

[8] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming, 2nd Ed.*, Brooks/Cole–Thomson Learning, Pacific Grove, CA, 2003.

[9] J. Kallrath, *Modeling Languages in Mathematical Optimization*, Kluwer Academic Publishers, 2004.

[1] For example, this is similar to the manner in which AMPL launches optimizers.

A Complete Python Implementation of the Simple Model

```
# Imports
from pyomo import *

# Setup the model
model = Model()

model.P = Set()

model.a = Param(index=model.P)
model.b = Param()
model.c = Param(index=model.P)
model.u = Param(index=model.P)

def X_bounds(j, model):
    return (0,model.u[j])
model.X = Var(index=model.P, bounds=X_bounds)

def Objective_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + model.c[j] * model.X[j]
    return ans
model.profit = Objective(rule=Objective_rule)

def Time_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + (1.0/model.a[j]) * model.X[j]
    return ans < model.b
model.Time = Constraint(rule=Time_rule)

print "ABSTRACT MODEL"
model.pprint()

# Create the model instance
instance = model.create("prod.dat")

print "MODEL INSTANCE"
instance.pprint()
```

DISTRIBUTION:

2	Regan Murray US EPA Facilities 26 W Martin Luther King Dr (MS 163) Cincinnati OH 45268	
1	MS 1316	Mark D. Rintoul, 1412
1	MS 1318	Robert D. Carr, 1412
1	MS 1318	Cynthia A. Phillips, 1412
1	MS 1318	Jean-Paul Watson, 1412
1	MS 1318	David Gay, 1411
1	MS 1318	Vitus Leung, 1415
1	MS 1318	Suzanne Rountree, 1415
1	MS 1318	David Womble, 1410
1	MS 1320	S. Scott Collis, 1414
1	MS 1320	Jonathan W. Berry, 1416
5	MS 1318	William E. Hart, 1412
2	MS 9018	Central Technical Files, 8944 (1 electronic, 1 hardcopy)
2	MS 0899	Technical Library, 9536 (1 electronic, 1 hardcopy)
1	MS 0123	D. Chavez, LDRD Office, 1011

Sandia National Laboratories