

# A General Integer-Programming-Based Framework for Sensor Placement in Municipal Water Networks

Jonathan Berry\*    William E. Hart<sup>†</sup>    Cynthia A. Phillips<sup>†</sup>  
James Uber<sup>‡</sup>

February 26, 2004

## Abstract

We present a new integer-programming formulation for sensor placement in municipal water systems. Berry, Fleischer, Hart, and Phillips [1] introduced a time-independent sensor-placement model. To avoid explicit references to time, this model assumed conservatively that consumers are protected only if every path from a contaminant introduction site is guarded by a sensor. The model also assumed that flow within the pipes was reasonably swift, if not quantified, so that flow patterns will not shift before a contaminant is detected.

We allow more realistic modeling of water transport by decoupling contaminant transportation from sensor placement decisions and explicitly considering time. We use discrete-event simulation, using velocity information derived from EPANET, to calculate contamination timing in the network for each of an enumerable number of attack scenarios. We then use an integer program to select a set of sensor locations that minimizes the release of contaminant across all these attack scenarios.

Initial computational experience on two real networks shows that the discrete-event simulator is very fast and the resulting integer programs are tractable for moderate-sized attack sets.

---

\*Computer Science Dept, Lafayette College, Easton, PA, 18012; PH (610)330-5115, [berryjw@cs.lafayette.edu](mailto:berryjw@cs.lafayette.edu).

<sup>†</sup>Algorithms and Discrete Math Dept, Sandia National Laboratories, Albuquerque, NM; PH (505)844-2217, (505)845-7296 [{wehart,caphill}@sandia.gov](mailto:{wehart,caphill}@sandia.gov). 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.

<sup>‡</sup>USEPA, Cincinnati, OH and Dept. of Civil Engineering, University of Cincinnati, Cincinnati, OH; PH (513)569-7974 [uber.jim@epa.gov](mailto:uber.jim@epa.gov).

# 1 Introduction

In this paper we introduce a new integer-programming-based model for sensor placement in municipal water systems to detect maliciously injected contaminants. We wish to place a limited number of perfect sensors in the pipes or junctions of a water network so as to minimize the expected amount of damage to the public before detection assuming the attack occurs on a “typical” day. This general model decouples the sensor placement problem from the water transport model. The decoupling allows researchers to experiment considerably with the transport model, gradually adding realism, while working with a consistent integer program model.

Integer programs are a class of optimization problem. Specifically, an integer program maximizes or minimizes a linear objective function subject to a set of linear constraints on the variables. Some of the variables must have integer values. Such variables usually represent a decision where a fractional value is impossible. In our model, a set of variables represent decisions to place a sensor at particular locations. Each decision has only two choices: yes (1) and no (0); one cannot partially install a sensor to achieve partial benefit. The introduction of this non-linearity makes the general problem formally intractable. However many integer programs can be solved to optimality by intelligent branch-and-bound methods using a commercial code like CPLEX from iLog or a free research code such as PICO[2].

We model a municipal water network as a graph  $G = (V, E)$ , where the vertex set  $V$  is a set of junctions, tanks, or locations of water consumption and the edge set  $E$  is a set of pipes. In higher-granularity models of a network, each node could represent an entire neighborhood or city region. We assume that water demands follow a small set of patterns, perhaps 4-24 per day. These patterns represent the demand during a particular time interval (say midnight to 2am) on a “typical” day. Because each pattern holds steady for one or more hours, we assume the gross water flow induced by these demands holds steady during the time period associated with that pattern.

We assume an attacker will contaminate the network at precisely one point through a single continuous injection (e.g. from a single full water tanker truck with a maximum capacity of 5500 gallons). Our model requires a risk profile, that is, a probability distribution that weights the likelihood of an attack at a particular point at a particular time of day. As the contaminant is injected into the system, it travels through the network according to the shifting velocities (directions and speeds) induced by the demand pattern at any given time of day. For example, contamination could enter a pipe, travel part way down, and then, with a shift in use pattern, it could reverse, never touching the far end of the pipe. Thus, the evolution of the plume of contamination could vary considerably depending upon the precise timing of the attack (for example, if it is at the start of a particular flow/demand pattern or at the end of it). Running a full EPANET simulation for every possible attack point and time could be prohibitively expensive. Because we assume flow patterns are stable for long periods, we replace the detailed hy-

draulic simulation with a much faster and simpler discrete event simulation. This simulation models the evolution of the contaminant plume and extracts the data necessary for the integer-programming model. We can replace this piece with a higher fidelity simulation without changing the integer program.

We wish to place a limited number of sensors in a water network to most effectively protect the city’s population. For this paper, we allow sensor placement on any node or in the geometric center of any edge. We can easily replace these assumptions with more a more general model of edge sensor placement. In practice, some locations may be too difficult to access. Therefore our model allows forbidden sensor location, though we did not exercise this option on our initial tests.

We assume these perfect sensors raise a general alarm precisely when passed by contamination of sufficient magnitude. Any contaminated water that leaves the system before this detection could cause harm. Ultimately, we wish to minimize the number of people exposed to a dangerous level contaminant across the attack profile. When the risk profile is a probability distribution, the objective is then to minimize expected dangerous exposure, as measured by the number of people consuming water at contaminated nodes. Though this is correlated with the volume of contaminant leaving the system, it is not necessarily directly related. Counterexamples include a manufacturing facility that consumes large amounts of water to cool equipment and a playground water fountain where a relatively tiny amount of consumption exposes a large, particularly vulnerable population. However, because of data issues that will be discussed in section 7, the preliminary computational experiments in this paper minimize the amount of contaminant released to the public before detection.

The most closely related work to our current model is Kessler, Ostfeld, Sinai, and Salomon’s set cover model to place a minimum number of sensors to insure detection before a given volume of contaminant is consumed [4, 5]. They consider net point-to-point flow rates based on EPANET simulations and assume that a point-to-point contaminant flow is detected if there is a sensor on the shortest path between these points. They solve their set cover problem using heuristics. Our IP has set cover structure, though each element has a different value for each set it covers and a set is uniquely covered by its most valuable element. [1] cites other related work.

The remainder of the paper is organized as follows. In Section 2, we discuss the water transportation model and the design of our discrete-event simulator. In Section 3 we describe our general integer-programming model. In Sections 4, 5 and 6 we describe our experimental design, data sets, and results respectively. Finally in Section 7 we discuss population modeling and give some conclusions.

## 2 Water Transport Model

In this section we describe the design of our discrete-event simulator DEWS (discrete-event water simulator). For each demand pattern, DEWS requires as input the edge flow velocities, determined by running EPANET until the flows stabilize; the demand for each node; and the time interval during which it holds.

It also requires a time threshold  $T$ . Given an (attack point, time) pair, DEWS computes the time each potential sensor location (node or edge midpoint) is first contaminated. Then for each possible sensor location  $\ell$  that is contaminated at time  $t_\ell \leq T$ , DEWS computes (and outputs) the total contaminant released from the network by time  $t_\ell$ . This is the contaminant that would be released to the public if a sensor at location  $\ell$  is the first to signal an alarm. DEWS also outputs the total volume consumed by the network during the entire attack duration. This is the amount consumed if the attack is never detected.

Contaminant travels through the network at pipe velocities as discrete balls, each tagged with a size. If flow in a pipe reverses while a ball is traveling down (due to a pattern switch), then the ball reverses direction at that time. We assume perfect mixing at nodes. We assume that for every non-tank node, water flow is conserved for each flow pattern (total volume rate in = total volume rate out + demand rate for each pattern). Tanks have storage capacity and can therefore support an unbalanced flow in either direction. Since we are only tracking contamination, all tanks are initially empty (of contaminants).

We model the attack as a single continuous 5500-gallon injection. We assume contaminant flows into the attacked node cause negligible perturbations to baseline network flow rates. Initial contaminant flow rate will depend upon the adversary’s pumping power, and hence his budget. According to our assumption, the contaminant cannot enter the system faster than the normal output rate from the attacked node. For this paper, we assume this worst-case pumping rate. DEWS models the injection as a release of balls of the appropriate size from the attack point every minute, possibly crossing flow pattern time boundaries, until the full 5500 gallons is released. Thus the attacker creates a temporary (one-way) tank at the attack point.

DEWS tracks the creation and flow of contaminant balls through the network as follows. There are two fundamental event types: the arrival of a ball at a node or edge and a change in flow pattern. To simplify the discussion, we consider edge midpoints to be demand-free degree-two nodes. The first event in the system is the arrival of a ball at the attack node. In general, when a ball hits a node, it is coalesced with any balls arriving at the same time, then divided among the outgoing edges and any local demand based on flow rate. For example, if there are two outgoing edges, each carrying 49% of the incoming flow and 2% of the incoming water is consumed, then DEWS combines all the contaminant balls arriving at the node into a single ball and then creates two balls, each with 49% of the original size. The 2% of the incoming ball that is consumed is removed from the system. If any of the newly-created balls has size below a specified threshold, then they are also removed from the system. DEWS then schedules an event for each of the newly-created balls: either the arrival at the endpoint of the edge it will traverse, or, if the ball will not arrive at the endpoint before the flow changes, an indication of the ball’s location at the next flow change somewhere partially down the pipe. If a tank node is storing during a particular flow pattern, it will accumulate contaminant volume as balls arrive (again, at a rate proportional to the storage rate). If a tank node is dispersing water, then we assume it disperses

pure contaminant (up to the amount it has stored), again at increments of one minute. Future versions of DEWS will reduce the amount returned to the system from tanks to account for dilution.

DEWS uses a priority queue. At each step it removes all events with a minimum time tag. This allows coalescing of balls at nodes (though we expect this may be a rare event in practice) and effectively implements a synchronization at flow-change times. DEWS records when a node or edge mid-point is hit by a ball for the first time.

The simulator stops when all contaminant has been removed from the system (via consumption or division to sizes below threshold) or after the time threshold  $T$ , whichever comes first. DEWS can enforce the time limit from the time of attack or from the time the first populated node is contaminated. In the latter case, we assume the contaminant has some health effects that will be detected within the simulation time limit, effectively acting like a sensor alarm.

### 3 The Sensor-Placement Integer Program

Given all the output from the discrete-event simulator, we determine the sensor placement with the following integer program.

The input data is as follows:

- $G = (V, E)$ , the network.  $V = v_1, \dots, v_n$  and  $E = e_1, \dots, e_m$ .
- $\alpha_{it}$ , the probability of an attack at node  $v_i$  at time  $t$ .  
Assuming exactly one location will be attacked sometime during the day, we have  $\sum_{(i,t) \in \mathcal{A}} \alpha_{it} = 1$ , where  $\mathcal{A} \subseteq V \times \tau$  is the set of attacks and  $\tau$  is the set of times used for at least one attack.
- $S_{\max}$ , the maximum number of sensors.
- $L \subseteq V \times E$ , the set of possible sensor locations.
- $L_a \subseteq V \times E$ , the set of network locations contaminated by attack  $a$ .
- $w_{aj}$ , for  $a = 1, \dots, |\mathcal{A}|$  and  $j \in L \cap L_a \cup \{q\}$ , where  $q$  is a dummy location; weights from the DEWS output.  
For each  $v \in L_a$ , let  $t_v$  be the time node  $v$  is contaminated by attack  $a$ . Let  $d_v(t_1, k)$  be the total demand at node  $v$  from time of day  $t_1$  through the next  $k$  time units ( $d_v(t_1, k) = 0$  if  $k \leq 0$ ). Recall that  $k$  can be longer than a day. The weight  $w_{aj}$  is the amount of contaminant consumed by the network before detection if a sensor at location  $j$  raises the alarm. That is, all potential sensor locations contaminated before  $t_j$  are not given a sensor. Thus we have  $w_{aj} = \sum_{v \in L_a} d_v(t_v, t_j - t_v)$  for all  $j \in L \cap L_a$ .  $w_{aq}$  is the amount consumed if no sensor raises an alarm:  $w_{aq} = \sum_{v \in L_a} d_v(t_v, T - t_v)$ .

The integer program (IP) uses the following variables:

- decision variable  $s_i$  for each potential sensor location  $i \in L$ . This variable is 1 if we place a sensor at location  $i$  and is 0 otherwise.
- derived variables  $b_{ai}$  for  $a \in \mathcal{A}$  and  $i \in L \cap L_a \cup \{q\}$ , where  $q$  is a dummy location. Variable  $b_{ai}$  is 1 if location  $i$  raised the alarm for attack  $a$ . That is, if location  $i$  is the first sensor hit for attack scenario  $a$ ; all better sensor locations are not given sensors. These variables need not have formal integrality constraints. They will always be binary in any optimal solution provided the  $s_i$  variables are binary. Omitting these unnecessary constraints can improve the practical performance of IP solvers.

For ease of notation, let  $\mathcal{L} = L \cap L_a \cup \{q\}$ , the set of useful sensor locations for attack  $a$  plus the dummy location. The IP to minimize consumption is:

$$\begin{aligned}
 \text{(MC)} \quad & \text{minimize} \quad \sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{L}} \alpha_a w_{ai} b_{ai} \\
 & \text{where} \quad \begin{cases} \sum_{i \in \mathcal{L}} b_{ai} = 1 & \forall a \in \mathcal{A} \\ b_{ai} \leq s_i & \forall a \in \mathcal{A}, i \in \mathcal{L} \\ \sum_{i \in \mathcal{L}} s_i \leq S_{\max} \end{cases}
 \end{aligned}$$

The first type of constraint assures that there is exactly one best sensor for each attack scenario. The second set enforces that a sensor cannot be best if it is never installed. The objective-function pressure then assures that the first eligible sensor in the list for attack  $a$  is chosen as best (barring zero-demand nodes). The last constraint enforces the limit on the total number of sensors. The objective minimizes the total consumption over all attacks (weighted by risk).

## 4 Methods

We have evaluated our sensor-placement strategy experimentally using two real networks. We ran EPANET to simulate a full day on each of the networks, once with 4 defined flow periods, and once with 24 flow periods. For each period, we extracted the flow rate in gallons per minute and velocity in feet per second along each edge. This information, along with the demands, serves as input to DEWS. We ran DEWS with a time threshold  $T = 72$  hours from the time of attack. DEWS produced the  $w_{ai}$  values for the IP, the total amount of contaminant consumed at all vertices from the beginning of the simulation of attack  $a$  to first detection (at location  $i$ ).

Our experiments on the datasets described below are divided into two categories. *Broad* experiments consider all vertices as potential attack points, but limit the number of attack times, and *focused* experiments consider all possible attack times (to a granularity of 15 minute intervals in the first simulated day), but a limited set of attack vertices. If the size of the dataset permits, a full (broad and focused) experiment can be run. The full version of this extended abstract will augment the preliminary results we present in Section 6 below.

Each attack involves an injection of 5500 gallons of contaminant (the storage capacity of a typical water truck) at the current rate of outflow from the attack vertex.

We use the AMPL modeling language [3] to formulate the integer program (MC). In all cases, we solved this IP using AMPL 9.0, which applied the CPLEX 9.0 IP solver. These tools ran on Linux workstations, and were able to formulate and solve each experiment we attempted in less than two hours. The IP solver never required more than 15 minutes.

## 5 Data

**Dataset 1** The first dataset, described in more detail as Dataset 3 in [1], was adapted from a local area network. This network has 470 nodes and 597 pipes<sup>1</sup>. This dataset is characterized by large variations in the number of reachable vertices from a given vertex.

With only 470 potential attack points and 96 potential attack times, we could run a full experiment on this dataset. We employed 100 sensors.

**Dataset 2** The second dataset was drawn from a moderate-sized city. There are 3647 nodes and 3803 pipes. With the quadratic combination of attack components, it was not feasible to run a full experiment on this dataset. Rather, we ran one suite each of broad and focused experiments. The selection of attack vertices or attack times was random, so we ran multiple trials of each experiment.

The broad experiment consisted of four potential attack times, drawn randomly from the 96 possible times. We allowed only 100 sensors to protect the network. The focused experiment allowed 200 potential attack vertices and the same number of sensors.

## 6 Results

The goal in writing our discrete-event simulator DEWS was to approximate longer, multiperiod EPANET simulations much faster. The average execution time for DEWS to simulate 72 hours on either data set was about a second. The largest running time was about a minute for attacks at nodes where flow was relatively fast and reached a large set of nodes. In comparison, an EPANET run for dataset 2 required 5 seconds to simulate 24 hours, one minute to simulate 48 hours, and failed to converge for a 72-hour simulation. Thus DEWS is normally at least 60 times faster than EPANET for this data set. Furthermore, there are obvious places within DEWS where some code optimization could offer considerable speed up, should it become necessary. This runtime improvement could be critical for obtaining a full data set for the IP model. For example, allowing an attack at any of 500 vertices during any of 100 different times of day requires 50,000 runs of DEWS. Even if each run takes roughly one second, as it does on a 3 Ghz modern machine with 2Gb RAM, the simulation phase takes almost a day of computing time. Assuming that the 500 vertices were randomly selected from a larger set and that we wish to generate at least 10 such trials in order to evaluate the sensitivity of the model, these runs would require roughly 10 days.

Table 1 presents the results of a single full experiment on dataset 1. The extremely low objective value, an expected value of only 0.57 gallons consumed

---

<sup>1</sup>The number of pipes given in [1] was 621, but it was later found that some of these pipes were closed off. These were deleted.

before detection, is an artifact of the node demand in this dataset. Over 250 of the 470 nodes have demand 0.0 for most flow periods, and another 100 vertices have demands of 0.01 gallons per minute.

Trial Number	Gallons consumed Before Detection	IP simplex iterations
1	0.57	379524

Table 1: The optimal solution for the full experiment on dataset 1 (470 potential attack points, 96 potential attack times) with 100 sensors and an attack of 5500 gallons of contaminant.

Table 2 presents the results of a *focused* experiments on dataset 2. For this extended abstract, we had insufficient time to generate statistically significant suites of experiments, so we show individual trials rather than attempting to computing statistics.

Trial Number	Gallons consumed Before Detection	IP simplex iterations
1	570.57	390409
2	443.32	332683
3	645.77	331922
4	656.20	376553
5	717.02	368058

Table 2: Summary of the values of optimal sensor configurations for dataset 2 with 200 potential attack points, 96 potential attack times, 100 sensors, and an attack of 5500 gallons of contaminant.

The results of our *broad* experiments on dataset 2 are presented in Table 3. For this dataset, the trend shows that placing sensors on roughly 2.6% of the possible sensor locations enables us to expect detection before roughly 32% of the contaminant has been consumed.

## 7 Extensions and Conclusions

As we discussed in the introduction, we would prefer to model direct population exposure in our IP objective. In this section, we discuss possible ways to model population exposure and consider model extensions and open questions.

In this model, a person is at risk if he or she consumes water from a node between the time it is first contaminated and the time of the first system wide detection. This could cover many time periods, even days. This makes estimation difficult since populations at nodes change over time and people move about the network, possibly receiving multiple exposures.

One possible way to obtain estimates of population exposure is to begin with population estimates for each node by time of day. Each node can have its own natural time divisions. For example, a mall could have one population while it is open and another while it is closed. There are two types of populations: *stable* populations such as families in homes or employees at a business, and *transient*

Trial Number	Gallons consumed Before Detection	IP simplex iterations
1	2508.25	343896
2	2467.37	369956
3	2489.84	345528
4	2488.84	373177
5	2429.35	373749

(a)

Trial Number	Gallons consumed Before Detection	IP simplex iterations
1	1840.73	271711
2	1806.19	286316
3	1825.88	266214
4	1832.99	291979
5	1778.44	289512

(b)

Table 3: Summary of the values of optimal sensor configurations for dataset 2 with 3647 potential attack points and 4 potential attack times, an attack of 5500 gallons of contaminant, and (a) 100 sensors, (b) 200 sensors.

populations, such as customers at a grocery store. We specify a stable population with a single number for each time interval. We specify a transient population with a *rate* (number of water consumers/hour). Most nodes will have only one type of population. For a node with a stable population, the number of people at risk from a contamination is the *maximum* of the population at any time between the initial contamination and the detection. For a node with a transient population, the number of people at risk is the *sum* of the people at the node. This is the rate times number of hours, summed over all distinct population-rate time periods, where hours might accumulate over multiple days.

Census data provides a reasonable estimate of stable residential population figures and one could use population statistics such as number of people who work outside the home to guess typical household schedules. There is no obvious source of information on transient populations, unless cities have run traffic surveys around malls, for example. Because there will be a large amount of uncertainty in the population data, we expect overcounting due to population mobility will be a low-order effect.

For the two data sets in this study, there we had no basis for deviating significantly from demand-proportional populations, and no basis for judging transient populations. In cases where the population is strongly correlated with demand, we expect the IP to have roughly the same practical computational requirements as the one we used.

Though this problem has combinatorial structure, we are justified in using an IP solver, even though it has worst-case exponential time complexity. The problem IP MC is solving (choosing at most  $S_{\max}$  sensors to minimize the total weight accumulated across a set of (sub)permutations), is strongly NP-hard and

provably not approximable within any polynomial factor.

This work suggests some algorithmic research. Is there a way to merge (or prune) sufficiently similar attacks so that the resulting (smaller) IP is a provably-good approximation to the original? Given the approximations likely necessary to specify the objective function coefficients, it seems wise to reduce computation and space requirements by (closely) approximating the optimal solution.

Because water networks are planar, we assume a small fixed number of cycling flows, and the weights on the objective function are highly structured, one may be able to compute the set of attacks ((location, time) pairs) to simulate rather than choosing times in a set stride.

Another algorithmic question is stopping criteria for DEWS. Optimally, when there is no obvious time threshold, we'd like to stop when we have proven no new nodes can be visited. It's likely to be difficult to determine this in practice, but there is some hope with this simplified model that we can detect sufficient conditions in some cases.

The simple DEWS model can be extended in a number of ways. For example, DEWS could enforce different thresholds for contamination than for detection, could model decay by reducing the total amount of contaminant in the system, and/or could model dispersal by allowing the balls to take on finite, possibly growing diameter.

## References

- [1] J. Berry, L. Fleischer, W.E. Hart, and C.A. Phillips. Sensor placement in municipal water networks. In *Proceedings of the World Water and Environmental Resources Conference*, 2003.
- [2] J. Eckstein, W.E. Hart, and C.A. Phillips. Pico: An object-oriented framework for parallel branch and bound. In D. Butnariu, Y. Censor, and S. Reich, editors, *Inherently Parallel Algorithms in Feasibility and Optimization and Their Applications*, pages 219–265. Elsevier Science Publishers, Amsterdam, The Netherlands, 2001.
- [3] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. boyd & fraser publishing company, 1993.
- [4] A. Kessler, A. Ostfeld, and G. Sinai. Detecting accidental contaminations in municipal water networks. *Journal of Water Resources Planning and Management*, pages 192–198, 1998.
- [5] A. Ostfeld and E. Salomons. An early warning detection system (EWDS) for drinking water distribution systems security. In *Proceedings of the World Water and Environmental Resources Conference*, 2003.