SAND2013-4155C

Improved Methods for Sensor Placement in Water Networks to Minimize Worst Case Impact

Michael Davis (Argonne National Laboratory)
Robert Janke(EPA)
Cynthia A. Phillips (Sandia National Laboratories)

Sponsored by the US Environmental Protection Agency (EPA) National Homeland Security Research Center (NHSRC)

The Sensor Placement Problem

Issue: Contamination released in a municipal water network

Goal: develop early warning system

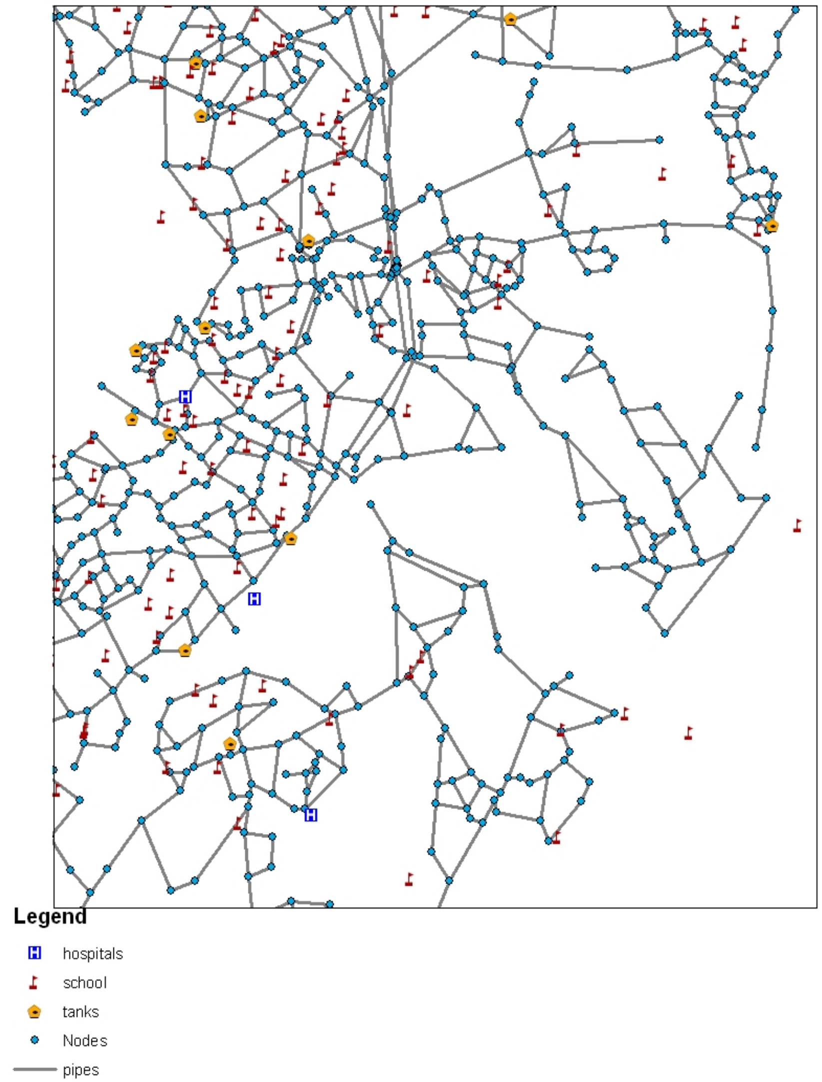
- Protect human populations
- Limit network remediation costs

Place sensors on

- Utility-owned infrastructure
- Schools
- hospitals

Sensors are expensive

- Cost of sensors
- Cost of installation



Modeling Assumptions

- Limited number of sensors (sensor budget)
  - Initially assume they are perfect
- Sensors raise a general alarm
  - Can model a response delay

- Fixed set of demand patterns for “typical” day
  - Seasonal variations
  - Special events
  - Weekday/weekend

Contaminant Transport Modeling

Water movement (direction, velocity in each pipe) determined by

- Demand (consumption)
- Pumps
- Gravity
- Valves
- Sources/tanks

Current (most trusted) simulator

- EPANET code computes hydraulic equations to determine flows
- Discrete-event simulation for contaminant movement

Modeling Events

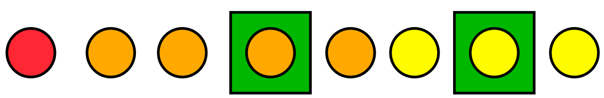
- Given: Set of events = (location, time) pairs
- Simulate the evolution of a contaminant plume
- For each event determine
  - Where/when event can be observed
  - Amount of damage prior to that observation

- Measures of damage/impact:
  - Population exposed
  - # deaths
  - Volume of contaminant release
  - Total pipe length contaminated
  - Time to detection
  - # failed detections

Witnessing an Event

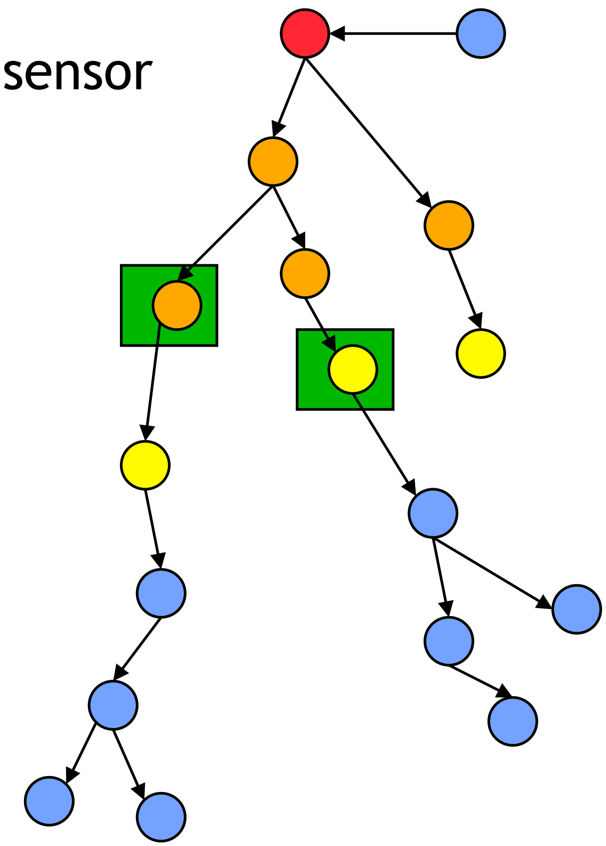
Simulator gives ordered list of nodes where a sensor could witness contamination

Witnesses:



This example has two (green) sensors.

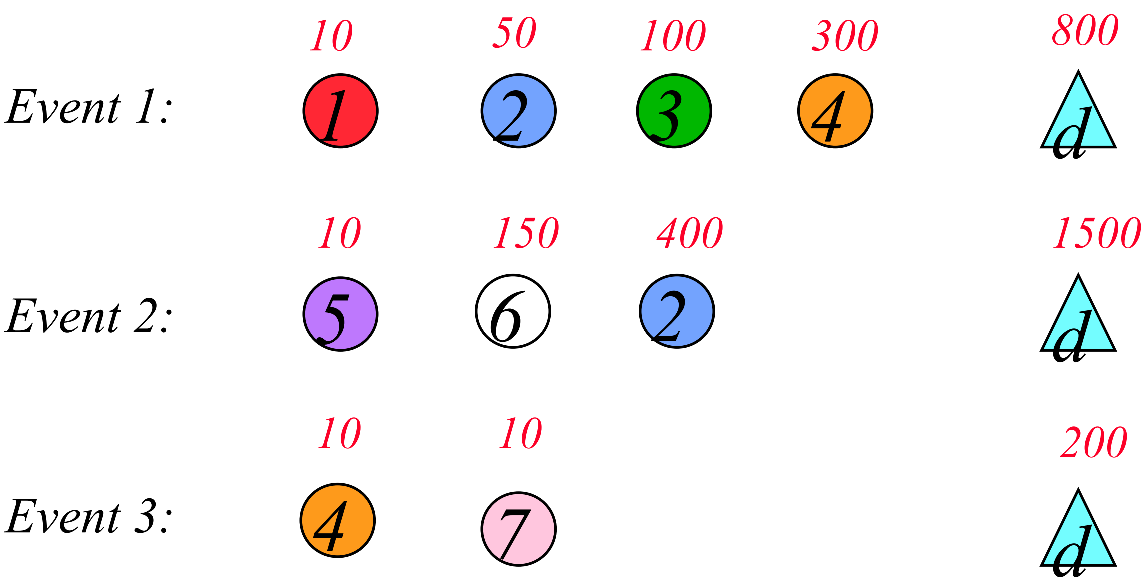
Perfect sensor model: first sensor in list detects the event.



Evaluating a Sensor Placement

- Impact in red

= dummy node (represents failure to detect)



Sandia National Laboratories

Evaluating a Sensor Placement

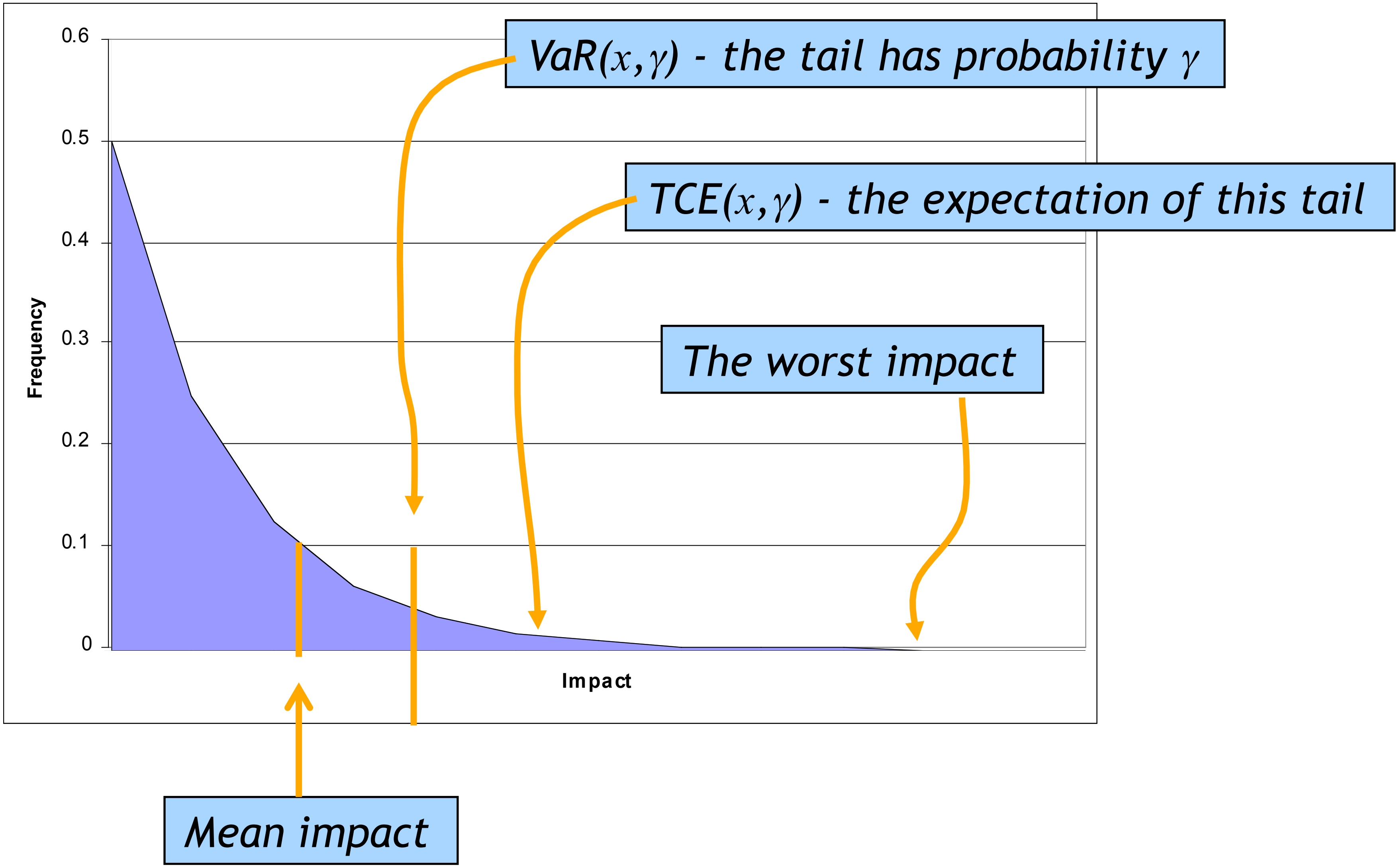
- Impact in red

d = dummy node (represents failure to detect)

						Impact:
	10	*50*	*100*	*300*	*800*	
Event 1:	1	2	3	4	*d*	*50*
	10	*150*	*400*		*1500*	
Event 2:	5	6	2		*d*	*400*
	10	*10*			*200*	
Event 3:	4	7			*d*	*200*

Choose sensors 2 and 3 (black)

Examples of Risk Measures



One Sensor Placement IP for Water Networks

Variables:

$$y_i = \begin{cases} 1 & \text{if we place a sensor at location } i \in \mathcal{L}, \\ 0 & \text{Otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if location } i \text{ raises the alarm (witnesses) event } j \\ 0 & \text{Otherwise} \end{cases}$$

Extreme points will have integer values for x_{ij} if the y_i are integral.

Each event has a dummy location to mark failure to detect

Sensor Placement Mixed Integer Program

$$\text{minimize} \sum_{j \in A} \sum_{i \in L_j} w_{ij} x_{ij}$$

$$s.t.$$

$$\sum_{i \in L_j} x_{ij} = 1 \qquad \forall j \in A \qquad \text{(every event witnessed)}$$

$$x_{ij} \leq y_i \qquad \forall j \in A, i \in \mathcal{L}_j \qquad \text{(need sensor to witness)}$$

$$\sum_{i \in L} y_i \leq p \qquad \text{(sensor count limit)}$$

$$y_i \in \{0,1\}$$

$$0 \leq x_{ij} \leq 1$$

For mean Sensor Placement = p-median

p-median problem:

- n possible facility locations
- m customers
- d_{ij} = distance from customer j to location i

- Pick p locations and assign each customer to an open location to minimize the total distance.

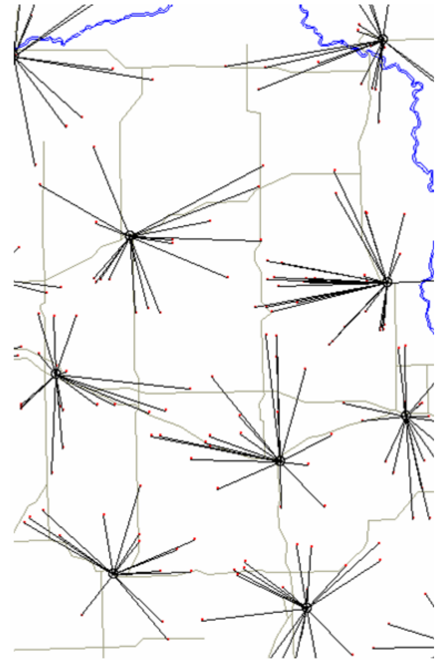
Sensor placement as a p-median problem:

- Sensors = Facilities
- Network locations = potential facility locations
- Events = Customers to be "served" (witnessed)
- "Distance" from an event j to a node i = impact if a sensor at node i witnesses event j.

Sensor Placement Heuristic Solvers

Grasp: Multistart local search

1. Build starting point
   - Add *p* sensors one at a time
   - Bias exponentially based on impact reduction
2. Greedy local descent
   - Neighborhood swaps sensor location with non-location



- Much faster than IP (sometimes 10x)
- Uses sparse matrix representation, but still requires superlinear space.

Almost always optimal

- Even with just one iteration of start + descent
- If not optimal, very close

Sensor placement to minimize worst case

- p-center instead of p-median

minimize W

$s.t.$

$\sum_{i \in L_j} x_{ij} = 1 \qquad \forall j \in A \qquad$ (every event witnessed)

$x_{ij} \le y_i \qquad \forall j \in A, i \in \mathcal{L}_j \qquad$ (need sensor to witness)

$\sum_{i \in L} y_i \le p \qquad$ (sensor count limit)

$\sum_{i \in L_j} w_{ij} x_{ij} \le W \qquad \forall j \in A \qquad$ (Each scenario obeys worst-case bound)

$y_i \in \{0,1\}$

$0 \le x_{ij} \le 1$

P-center GRASP heuristic

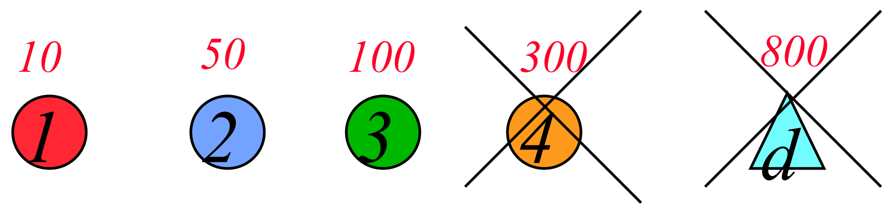
- Can do the same local search but use worst-case objective
- Loses some data structure/algorithmic optimizations
- Much slower than mean
- Can have much larger errors (10%+)

- Utilities, EPA, some researchers, etc, so used to having (heavily used) mean heuristic be optimal
  - Assumed would hold for other objectives too
  - Of course no reason to expect that (good behavior for mean still unexplained)

Change constraint for Objective

Find minimum number of sensors to achieve worst case impact W

- For each scenario, remove all witnesses with impact > W

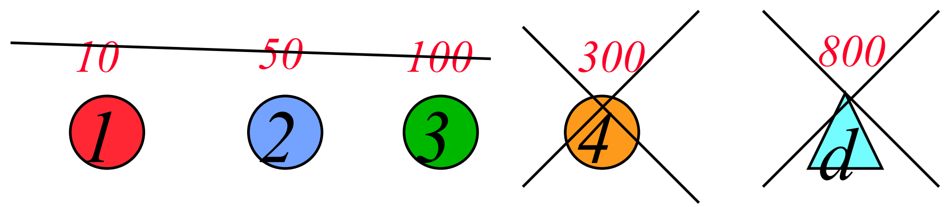


  - If none left, W infeasible for any # sensors

Change constraint for Objective

Find minimum number of sensors to achieve worst case impact W

- For each scenario, remove all witnesses with impact > W



  - If none left, W infeasible for any # sensors
- Ignore impact value on remaining locations, just a set

Set Cover:

- A set of sensor locations covers a scenario if at least one feasible location selected
- Find the smallest covering set using IP

Binary Search

- Put all impact values (from all events) in sorted array
  - No duplicate values
  - Binary search on array
  - No numerical tolerance/convergence issues
- Starting bounds
  - Run p-median heuristic (gives placement X_m)
  - Worst case impact over all events for X_m is upper bound
  - Value of optimal mean is lower bound
    - Heuristic usually optimal enough (check)
  - Have not verified initial bounding is a win

- Implemented in python. Creating the array (reading impact file) can be very expensive. C++ for future maybe.

Real Networks for Experiments

- 6 Networks
- Heuristic depends mostly on # nodes
- Binary search on size of impact file and # iterations

Name	# nodes	# contamination events	#impacts (varies)	impact file size
Net2Morph	3358	1621	1.2M	22M
NetI	6809	6671	4.7M	82M
BWSN	12527	10552	8.2M	156M
NetE	13634	8679	57M	1G
NetB	42698	28675	36M	744M
NetN	48164	9162	38M	772M

Sample Results

Name	# heuristic value	heuristic runtime	binary search value (opt)	binary search runtime	% error
Net2Morph (best)	861	1131s	852	158s	1
Net2Morph (worst)	1166	778s	1059	204s	10
NetI (best)	897	16072s	890	694s	0.8
NetI (worst)	570	5731s	518	473s	12
BWSN (best)	1037	33040s	1037	239s	0
BWSN (worst)	1092	26339s	980	216s	11
NetE	1070	47792s	1025	4650s	4.4
NetB	8472	666622s	8320	19360s	1.8
NetN	7286	358697s	6851	21282s	6.3

- IP: cplex 12.4 (parallel) still had > 40% gap for Net2Morph after 13 hours of wall clock time.

Binary Search Runtime Breakdown

- Moving from python to C++ will significantly reduce time to read impact file and create the impact array
- All times in seconds
- Set cover IPs solved with PICO (open-source solver)

Name	compute bounds	create array	# iterations	all search iteration time	total
Net2Morph (best)	31	98	10	30	158
Net2Morph (worst)	29	142	11	33	204
NetI (best)	215	338	11	140	694
NetI (worst)	203	134	11	135	473
BWSN	589	1504	11	232	2325
NetE	742	2141	11	1766	4650
NetB	6344	10760	16	2256	19360
NetN	2743	17096	13	1443	21282

Constrained mean

- Optimize mean subject to constraint on worst case
- Simply remove impacts that violate worst-case constraint
- Can still use simple mean heuristic
  - Compute best feasible worst case to insure feasibility
- Generally can achieve near-optimal mean within 5% of best worst

	Net2Morph	NetI	BWSN
opt mean	(205.18, 1458)	(33.38, 1387)	(64.01, 1490)
opt worst	(245.79, 1059)	(42.51, 518)	(89.63, 1049)
constrained mean	221.38	38.11	73.69
worst relaxed 5%	(208.33, 1107)	(37.85, 535)	(66.14, 1085)
worst relaxed 10%	(208.16, 1149)	(37.85, 535)	(66.14, 1085)

Value at Risk (VaR)

- δ = percent of scenarios that have impact greater than VaR
- δ usually small (e.g. 0.05)

Can use binary search over median

- Guess value V_g
- If impact < V_g, set impact to 0
- If impact > V_g, set impact to 1
  - Perturb V_g so not equal to any impacts (+ or – epsilon)
- If optimal mean > δ x (# scenarios), V_g too high, else too low
- Stop when V_g is bounded above and below by adjacent (or same) impact values

Conclusions

- Simple neighborhood swap heuristic is not optimal for worst-case objective
  - Should not be trusted the way it is trusted for p-median
- Binary-searched-based solver gives optimal solution much faster
  - Will be made available in EPA's Water Security Toolkit (WST).