Contamination Status Algorithm (CSA)
User's Manual (Version 1.0)

Annamaria E. De Sanctis, James G. Uber
Department of Civil and Environmental Engineering
University of Cincinnati, P.O. Box 210071, Cincinnati, Ohio 45221-0071

October 2, 2009

1 Introduction

The development of water quality sensors and sensor technologies makes it feasible to establish water quality monitoring networks in drinking water distribution systems. The collected water quality data can be used to determine the location and time of the source contamination in the case of terrorist attack or accidental contamination intrusion. If the contaminant reaction can be modeled reliably within the pipe network and the sensors can measure water quality quantitatively, minimization of the difference between modeled and measured water quality is one approach to solution of the contaminant source determination problem. The underling mathematical problem is, however, inherently ill-posed, due to the shortage of measurements compared to source parameters, and regularization methods are required to force identification of a unique solution. Further, it is usually the case that the contaminant reaction dynamics are unknown, and/or the sensor can only detect the presence or absence of the contaminant and not the quantitative concentration. Even if the source determination problem can be formulated mathematically and optimization algorithms can be applied to solve the problem, the decision variable dimension can be huge since contamination can happen anywhere and anytime in the network. An alternative practical method is developed by De Sanctis et al. (2006, 2009) to identify all possible locations and times that explain contamination incidents detected by the water quality sensors. It is assumed that contaminant injections occur at network junctions and over discrete time intervals. The method only requires the positive/negative sensor status over time, and knowledge of network hydraulics. A particle backtracking algorithm is used to identify the water flow paths leading to each sensor measurement and the travel time from the junctions along the flow paths to the measurement. Those locations and times that are connected to positive sensor measurements and are not connected to negative measurements are the possible sources, assuming no false positive/negative readings and an accurate hydraulic model. This method can also be used

as a pruning step for solving the source identification problem using optimization algorithms, as it reduces the number of decision variables by eliminating locations and times that are inconsistent with the sensor responses. The method also forms the basis for incorporating important concerns about hydraulic and sensor uncertainty, which are likely to enlarge the set of possible sources.

2 Overview of Contamination Status Algorithm

The *Contamination Status Algorithm* (*CSA*), De Sanctis et al. (2009), is a method to identify all the possible contamination sources in space and time. It updates the contamination possibility status of all nodes and time intervals using the network flow paths that deliver water to water quality monitoring stations. These monitoring stations are assumed to detect the presence/absence of contamination in a water distribution system. The network flow paths for each station are determined using the *particle backtracking algorithm* (Shang et al., 2002), as implemented in *EPANET-BTX*, a pre-release version of the EPANET backtracking extension that uses the EPANET hydraulic network model.

In this section we briefly review the particle backtracking algorithm that forms the basis for the source identification approach, and introduce key notation and concepts. Zierolf et al. (1998) and Boccelli et al. (1998) showed that the output node concentration in a network with time varying flows is a linear superposition of the history of mass additions at "upstream" input nodes. Their results - exact for linear reaction kinetics - can be summarized,

$$C_j(t) = \sum_{(i,\tau)\in U_j(t)} I_{ij}^{\tau}(t) u_i^{\tau}, \tag{1}$$

where $C_j(t)$ is contaminant concentration at output node j and time t, u_i^{τ} is contaminant source strength at input node i during time interval τ, and the *impact coefficient* $I_{ij}^{\tau}(t)$ is the concentration response at output node j and time t, to a unit source strength addition at input node i during time interval τ. The $U_j(t)$ are *upstream reachability sets*; these hold the upstream node-interval (i, τ) pairs that are connected to node j at time t, given the network topology and known history of time varying flow rates. Each $(i, \tau) \in U_j(t)$ is connected to downstream output node j at time t in the sense that source strength u_i^{τ} has a non-zero impact on concentration $C_j(t)$; all other (i, τ) pairs are disconnected from node j at time t, and $I_{ij}^{\tau}(t) = 0$. The magnitude of $I_{ij}^{\tau}(t)$ quantifies the strengths of the hydraulic connections between upstream and output nodes, separated by a time delay. We use these impact coefficients in the CSA algorithm described below to determine, given a positive monitoring state at the output node, the possible source locations and historical times. Similarly, we use them to determine, for a negative monitoring state, the source locations and historical times that are not possible, and that *must be excluded*.

Computation of the upstream reachability sets and the impact coefficients is performed by the particle backtracking algorithm (Shang et al., 2002). For any output node j and

time t, the reachability sets, $U_j(t)$, and the corresponding impact coefficients, $I_{ij}^{\tau}(t)$, can be computed by tracking water parcels in reverse time from the output node as they propagate upstream to different potential contaminant sources. When a junction node is reached, new subparcels are created according to the inflows and the same process continues along different paths. By making explicit the space-time relationships between input source strength and output sensor concentrations, the reachability sets allow efficient pruning of the parameter space, thus reducing problem size and imposing an *a priori*, physically based, "structural regularization" on the solution. Without such pruning, the number of unknown source strengths is on the order of the number of network nodes multiplied by the number of source time intervals – easily exceeding 10^6 unknown parameters for practical problems. The CSA algorithm presented in the following section attempts to recover the locations and time signatures of contamination source strengths, using eq. (1) to relate the output (sensor) and input (source) concentrations. Practical solution of such a problem requires that the potential input source strengths be discretized in space and time. Eq. (1) tacitly assumes that contamination sources are only considered at the nodes of the hydraulic network, and that the temporal signature of the source strengths at any node is discretized into the analysis time intervals indexed by τ. For a real-time algorithm, it is inconvenient to reference the time intervals to an arbitrary initial condition, because then as time increases the range of indexed time intervals must also increase. Thus we divide time, and the source strengths, into K equal intervals of duration δt, but index them backwards in time, such that the first interval is that which contains the output time t, the second interval immediately precedes the first, and so on. By selecting K such that $K\delta t$ exceeds the longest possible network travel path, we can march forward in time by steps δt, and still index the time intervals using $1 \leq \tau \leq K$. The units of source strength u_i^{τ} depend on the source type; a *set point booster* source that fixes the concentration of flow leaving the node has concentration units (M/L^3), whereas a *mass booster* source that adds a specified mass rate has mass rate units (M/T). Accordingly, the units of the impact coefficients $I_{ij}^{\tau}(t)$ are dimensionless for a set point booster source (in which case $0 \leq I_{ij}^{\tau}(t) \leq 1$), and (T/L^3) for a mass booster source $(0 \leq I_{ij}^{\tau}(t))$. While practically useful, the mathematical distinction between these two source types is trivial, since they are linearly related by the total flow through the source node.

To apply the CSA algorithm we assume that the network hydraulics are known; the contamination source strength is continuous over an analysis interval δt; each network junction is a potential source of contamination, and contamination can occur simultaneously at any number of nodes and analysis intervals (not necessarily contiguous); the M monitoring stations sample water quality at a subset of network junctions at arbitrary sampling times, and associated event detection algorithms calculate a binary event status: positive (contamination event detected), or negative (non-detect). Monitoring station sensors and event detection algorithms are assumed to accurately describe the water quality status, and to continue their sampling and data processing functions throughout the duration of an event (e.g., a positive status may be followed by a negative status at the same monitoring station, if the contamination event has passed the sensor).

The CSA iteratively processes sequential sets of real-time monitoring results (positive/negative status), polled during analysis time intervals of duration δt. During each new analysis time interval, monitoring results will in general have varying sampling times, due to random factors that affect the polling frequency in real-time communications systems. If monitoring station j has sampling time t during the most recent analysis time interval, the EPANET-BTX is used to calculate the upstream reachability sets $(i, \tau) \in U_j(t)$, and the corresponding impact coefficient $I_{ij}^{\tau}(t)$, as described above. The EPANET-BTX analyses are used to update an *(N x K) total source status matrix, S,* during each analysis interval. The *ith* row and τth column of S codes the status of the potential source at node i during the τth most recent analysis interval, as one of the following mutually exclusive states: *undafe* – possible contamination source, *safe* – not a possible contamination source, and *unknown* – insufficient information to classify the source as candidate or safe. The unknown state would occur whenever water from a candidate source does not pass any sensor location up through the current sampling interval.

Whereas S reflects contamination status and flow path information using all monitoring stations during the most recent K analysis intervals, we define an analogous station source status matrix, S_j, that reflects status information from monitoring station j at sampling time t during the current analysis interval. The information in the matrices S_j will be used to update information in S during each analysis time interval. When $I_{ij}^{\tau}(t) \geq \epsilon$ the cell of S_j is *safe* or *unsafe* whether the monitoring status is negative or positive respectively, otherwise the status is *unknown*.

The parameter $\epsilon \geq 0$ is used as a threshold to discriminate between weak and significant candidate sources. Given the station source status matrices, S_j, for each monitoring station sampled during the current analysis interval, it remains to describe how they are used to update the total source status matrix S, during each interval.

The CSA updating rule can be summarized simply:*safe* dominates *unsafe*, which dominates *unknown*. Thus if the current state is *unsafe*, it can only change to *safe*, and then only if a station state is *safe*. If the current total state is *safe* then the updated state will always be *safe*, while if it is *unknown* then the updated state is governed by the station state. The logic behind this updating rule is that an *unsafe* state indicates only the possibility of a source (one of many, in general), and can thus be overridden, whereas a *safe* state indicates that at least one significant path was connected to a negative contamination state, and thus must not be a source by our current assumptions of known hydraulics and accurate water quality status.

3 Program Usage

Table 1 lists in alphabetic order all the functions of the CSA Toolkit and their brief description. CSA is distributed as a compressed archive file named csa.zip which contains the following files:

- csa.pdf: CSA users manual.
- csa.dll: CSA dynamic linkage library
- csa.lib: Microsoft C/C++ library file for csa.dll.
- csa.h: C/C++ header file for CSA.
- epanetbtx.dll: EPANET-BTX dynamic linkage library that should be used in conjunction with the EPANET Toolkit library (epanet2.dll).
- epanetbtx.lib: Microsoft C/C++ library file for epanetbtx.dll.
- epanetbtx.h: C/C++ header file for EPANET-BTX.

Table 1: CSA toolkit functions

Function Name	**Purpose**
CSAaddsensor	Adds all sensor locations
CSAsetbinary	Sets the binary status for the measurements from the sensors
CSAcandidates	Provides the total number of unsafe and safe candidates identified
CSAclose	Closes the CSA toolkit system
CSAhydraulics	Runs the hydraulic simulation and prepares for back-tracking modeling
CSAopen	Opens the CSA toolkit system
CSAupdate	Identifies the status of all the node and time pairs being safe, unsafe, and unknown candidates

3.1 CSA API Illustration

CSA function library has to be used in conjunction with EPANET Programmer's Toolkit. The EPANET-BTX Programmer's Toolkit functions used by CSA are directly incorporated into the CSA Toolkit. CSA uses the network model data from the EPANET input file, therefore the following program example begins with the ENopen function to open an EPANET input file and read the network data (inpfile="net_true.inp").

```
#include "csa.h"
#include "epanet2.h"
#include "epanetbtx.h"
```

```
int main()
{
int errcode, historical-frame=24, sourcetype = EN_MASS;
int nsensors=3;
char * eparptfile = "rpt";
char * inpfile = "net_true.inp"; (network input file)
double ϵ=1.0e-10;
float sourcestep=3600.0, samplestep=3600.00;
errcode = ENopen(inpfile, eparptfile, "");
errcode = CSAopen(ϵ, sourcestep, historical-frame);
errcode = CSAhydraulics(sourcetype);
errcode = CSAaddsensor(nsensors, "sensorID.txt");
errcode = CSAsetbinary(samplestep, "sm.txt");
errcode = CSAupdate();
errcode = CSAcandidates();
errcode = CSAclose();
errcode = ENclose();
return 0;
}
```

The function CSAopen is used to start the CSA toolkit and associate the network data. It has three arguments: ϵ is the threshold to cut the flow paths that are not significant and it is set equal to 10^{-10}; sourcestep is the time step in seconds for the impact coefficient aggregation corresponding to time interval of the contamination source status matrix S; historical-frame is in this example 24 hours and indicates that from each current analysis time CSA determines the source contamination status of a network node for the last 24 hours. This parameter can be changed based on the number of information the user wants to store. The function CSAhydraulics is used first to run the hydraulic simulation and save the hydraulics data using EPANET Toolkit, second to associate the hydraulics solution with the backtracking modeling, and third to prepare CSA for the backtrack flow path analysis setting all the network nodes as possible contamination sources. The function CSAhydraulics has one argument: sourcetype refers to two of the four water quality type options used in EPANET. There are only two options since for a contamination event the source can be model either as mass booster or as flow-paced booster.

The function CSAaddsensor adds all the sensor locations selected by the user among the network nodes. The sensor locations are used later by the CSAsetbinary and CSAupdate. The function CSAaddsensor has two arguments: nsensors is the total number of sensors; "sensorID.txt" is a text file containing all the node IDs for the sensor locations. In this pseudocode example, there are three sensor locations sampling every hour (samplestep) and the number of sampling times is equal to the simulation duration diveded by the sampling time step. So the CSA will run for the entire simulation duration given by the Epanet input

file. At each sampling time the user sets the measurements from all the sensors by means of the function CSAsetbinary. The latest function, in fact, has as second argument the text file (sm.txt) of all the measurements from all sensors for the entire simulation duration.

The previous functions set all the input data for the source identification analysis, while the core of the contamination source identification toolkit is the function CSAupdate. It iteratively processes sequential sets of monitoring results (CSAsetbinary) pulled during the analysis time interval of duration sourcestep (3600 seconds in the example). CSAupdate uses EPANET-BTX to calculate the flow paths between the sensor location and the possible contamination sources. Then, it updates the contamination source status (safe, unsafe, or unknown) of each network node at each analysis interval during the historical analysis time frame (historical-frame is 24 hours).

In order to retrieve the results from CSAupdate, the user can choose to call one or more functions: CSAout, CSAcandidates, and CSAlocations. In the current toolkit version CSAout and CSAlocations are not available, but the authors envisioned a more detailed version of the CSA toolkit that will be released next. In the example, at each new analysis time CSAcandidates returns the number of node-time pairs identified as safe candidates and unsafe candidates during the last 24 hours (historical-frame). The outputs of this function are directly store in a text file ("candidates.txt"). Further the function CSAupdate returns a more general output: a text file "Smatrix.txt" that shows at each analysis time the Contamination Source Matrix S from the first analysis time to the end of the simulation duration.

The code concludes with the functions CSAclose and ENclose to free the memories used by the CSA application and EPANET. Note that to run the following code you need 3 additional files as described before: the network input file, sensor IDs file, sensor measurements file.

4 Application Programming Interface of CSA

4.1 CSAopen

`int CSAopen(double` ϵ`, float sourcestep, int historical-frame)`

1. *Description*
   Opens the CSA toolkit system. Allocates memory data for CSA modeling. Allocates memory for contamination source status matrix S for all network nodes and for the length of simulation (S is initialized to unknown status).

2. *Arguments*
   ϵ: threshold for the significant flow path (total impact coefficient $I_{ij}^{\tau}(t) \geq \epsilon$);
   sourcestep: time step (τ)to integrate impact coefficients ($I_{i,j}^{\tau}$), in the unit of seconds;
   historical-frame: historical analysis time frame (number of hours).

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*
   Use CSAopen only after opening the network model input file with ENopen(”*.inp”,”*rpt”, ””).

4.2 CSAhydraulics

`int CSAhydraulics(int sourcetype)`

1. *Description*
   Runs hydraulic simulation using EPANET2 and opens the backtracking toolkit, reads the hydraulic results, initializes data for backtracking modeling and every network node as a possible source.

2. *Arguments*
   sourcetype: water quality type(EN_MASS or EN_FLOWPACED as defined in EPANET2).

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*

4.3 CSAaddsensor

`int CSAaddsensor(int nsensors, char *sensorfile)`

1. *Description*
   Adds all the sensor locations selected among the network nodes.

2. *Arguments*
   nsensors: total number of sensor locations;
   sensorfile: text file with the sensors IDs.

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*

4.4 CSAsetbinary

`int CSAsetbinary(float samplestep, char * measurementfile)`

1. *Description*
   Sets new monitoring station status (positive/negative) and associated sampling time. This function is used to collect samples to be analyzed during the current analysis time interval of duration $sourcestep \geq samplestep$.

2. *Arguments*
   samplestep: sampling time interval (in unit of second);
   measurementfile: text file containing the sensor measurements from all the sensor locations set in CSAaddsensor.

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*
   This is called before CSAupdate.

4.5 CSAupdate

`int CSAupdate()`

1. *Description*
   Runs the particle backtracking simulation from each sensor location at a particular output time (based on the sampling time). Applies the update CSA rule for all the network nodes and for an historical analysis time frame equal to historical-frame using the information from all the sensors at the current analysis time. Finally at each analysis time computes the Contamination Status Matrix S.

2. *Arguments*

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*
   This function is called after CSAbinary, because it reads sensor measurements from all the sensor locations. Gets impact vector for each possible source node at the current analysis time. The CSAupdate returns an output as a text file Smatrix.txt. This file contains all the Contamination Source matrices S computed during the simulation duration; each matrix corresponds to a particular analysis time. The matrix S has

number of rows equal to the total number of nodes; the number of columns increases until it is equal to the historical analysis time frame /source time step. In the example in section 3.1 the max number of colums of matrix S is $24 = \frac{historical-frame*3600}{sourcestep}$

4.6 CSAcandidates

`int CSAcandidates()`

1. *Description*
   Computes the total number of unsafe candidates and safe candidates for each analysis time processed by CSAupdate. Returns a text file called "candidates.txt"; writes for each analysis time the number of unsafe and safe candidates identified in the historical analysis time frame.

2. *Arguments*

3. *Returns*
   Returns an error code if there are errors, otherwise returns 0.

4. *Notes*
   This function is called after CSAupdate.

4.7 CSAclose

`int CSAclose()`

1. *Description*
   Closes the CSA toolkit system. Frees the memory used by CSA, closes the BTX application.

2. *Returns*
   Returns 0.

5 CSA Error Codes

References

Boccelli, D., M. Tryby, J. Uber, L. Rossman, M. Zierolf, and M. Polycarpou (1998). Optimal scheduling of booster disinfection in water distribution systems. *Journal of Water Resources Planning and Management 124*(2), 99–110.

De Sanctis, A., F. Shang, and J. Uber (2006). Determining possible contaminant sources through flow path analysis. In *Proceedings of the 8th Water Distribution System Analysis Symposium*, Cincinnati, OH.

De Sanctis, A., F. Shang, and J. Uber (2009). Real-time identification of possible contamination sources using network backtracking methods. *Accepted by Journal of Water Resources Planning and Management, ASCE*.

Shang, F., J. Uber, and M. Polycarpou (2002, May). Particle backtracking algorithm for water distribution system analysis. *J. Environ. Eng. 128*(5), 441–450.

Zierolf, M. L., M. M. Polycarpou, and J. G. Uber (1998, July). Development and autocalibration of an input-output model of chlorine transport in drinking water distribution systems. *IEEE Trans. on Control Systems Technology 6*(4), 543–553.