SANDIA REPORT

SAND2015-10401
Unlimited Release
Printed November 2015

ALADDIN: The Automatic Loader of Accident Data for Dynamic Inferencing Networks

Michael C. Darling, Thomas B. Jones, George F. Luger, Katrina M. Groth

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

Sandia National Laboratories

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

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

SAND2015-10401
Unlimited Release
Printed November 2015

ALADDIN: The Automatic Loader of Accident Data for Dynamic Inferencing Networks

Michael C. Darling, Katrina M. Groth
Risk & Reliability Analysis
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-MS0748

Thomas B. Jones, George F. Luger
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131

Abstract

Sandia has initiated the development of a methodology to generate "SMART (Safely Managing Accidental Reactor Transients) Procedures" to support nuclear power plant operators in diagnosis of severe accidents. The theoretical framework for developing SMART procedures involves coupling dynamic PRA, nuclear reactor simulation codes, and Bayesian Networks (BNs) provided by the University of Pittsburgh's Structural Modeling, Inference, and Learning Engine (SMILE) and its graphical inferencing tool (GeNIe) to provide fast-running diagnostic support.

A critical challenge when dealing with the output from dynamic PRA simulations is that each accident simulation produces hundreds of megabytes of data over thousands of time steps. Only a fraction of this information can be included in models designed for faster-than-accident-time use. In order for the SMART Procedures concept to be scalable to nontrivial diagnosis scenarios, it became essential to develop a method to automate the handling of the inputs to the BN. Therefore, Sandia partnered with the University of New Mexico to develop a system that would automate the processing of simulation data for input into a BN model.

This report outlines the development of the Automatic Loader of Accident Data for a Dynamic Inferencing Network (ALADDIN). ALADDIN automates the process of parsing and analyzing reactor simulation data to be input into GeNIe BNs. This was accomplished by creating a Java-based program which processes and discretizes the raw simulation data, calculates probability tables for the states of the plant parameters conditioned on the reactor states, and uses these calculations to build a bayesian network using the SMILE framework. Additionally, the system calculates the relative information gain value of each plant parameter. These calculations provide insight into which parameters would be most pertinent during particular accident scenarios. The initial prototype of ALADDIN outputs a diagnostic BN model for two accident types and twelve reactor parameters in a sodium fast reactor design.

Contents

Figures

Tables

Nomenclature

ALADDIN	Automatic Loader of Accident Data for a Dynamic Inferencing Network
BN	Bayesian Network
DBN	Dynamic Bayesian Network
DDET	Discrete Dynamic Event Tree
GeNIe	Graphical Network Interface to SMILE
KL	Kullback-Leibler
LWR	Light Water Reactor
PRA	Probabilistic Risk Assessment
SAMG	Severe Accident Management Guideline
SFR	Sodium Fast Reactor
SMILE	Structural Modeling, Inference, and Learning Engine developed by the University of Pittsburgh's Decision Systems Laboratory

1 Introduction

Sandia has initiated the development of a methodology to generate "SMART (Safely Managing Accidental Reactor Transients) Procedures" to support operators when diagnosing severe accidents in nuclear power plants. The theoretical framework for developing SMART procedures involves coupling dynamic PRA, system simulation codes, and Bayesian Networks (BNs) to provide fast-running diagnostic support [1]. The methodology takes outputs from dynamic PRA[1] and aggregates them into a BN which can then be used to provide decision support using native features in the graphical network interface, GeNIe, for the SMILE framework [4]. In short, the method uses a combination of tools to generate accident scenarios and simulate plant response; the results are fed into a BN model to enable assisted diagnosis. This approach provides a process for extensive and comprehensive modeling of both the accident space and the plant response, in a fast-running framework.

Dynamic, simulation-based Probabilistic Risk Assessment (PRA) methods can provide a scientific basis for supporting the diagnosis and response planning for current and future reactor designs. Recent advances in computing enable simulation-based PRA approaches to explore thousands of accident scenarios. Coupling these scenarios with plant simulations allows prediction of plant parameters and consequences associated with each accident scenario. In effect, running thousands of advanced PRA simulations allows experts to explicitly map out the relationship between known accident scenarios and observable reactor parameters. Dynamic PRA offers a comprehensive understanding of the accident scenarios and their associated plant states.

SNL developed a proof-of-concept BN using a Sodium Fast Reactor (SFR) model with 6 nodes using data from 84 simulations [5]. The initial BN model illustrated a promising concept, however, a key challenge in furthering its development was that the output from the Dynamic PRA simulations produces hundreds of megabytes of data over thousands of time steps. In order to develop a scalable system able to model non-trivial diagnosis scenarios, it became necessary to automate data processing and BN model construction. It also became necessary to discretize the dynamic PRA data since it is too large for BN models designed for faster-than-real-time analysis: the BN model must process accident diagnostics in a constrained amount of time.

The prototype presented in this report is the next iteration of the initial model developed by Groth et al. in 2014 [5]. The data set used for this study contains approximately $7,000$ accident progressions which constitute more than 280 megabytes of data. The system measures all of the parameters for each simulation and stores it in an efficient data structure which allows for fast access. This report outlines the development of the Automatic Loader of Accident Data for a Dynamic Inferencing Network (ALADDIN).

ALADDIN represents a major advance in the SMART procedures framework as it automates the process of parsing and transferring reactor simulation data into GeNIE BNs. This allows for scability in model size: whereas the previous hand-quantified model took one week to build, ALADDIN builds BN models in a matter of minutes. Additionally, ALADDIN uses information

[1]Discrete Dynamic Event Trees (DDETs) coupled to a nuclear reactor accident simulation code (e.g., MELCOR[2] or SAS4a[3])

theory to calculate the relative pertinence of the plant parameters. ALADDIN contributes significantly to the tools within the SMART procedures framework for supporting operators during beyond design basis accidents.

1.1 Structure of a "SMART Procedures" Dynamic Bayesian Network

The goal of the Dynamic Bayesian Network (DBN) is to learn the values of the plant parameters given the states of the reactor systems. Figure 1 illustrates a dynamic conceptualization of the Transient Overpower (TOP) and Loss of Flow (LOF) diagnosis problem. This figure contains a plate-based dynamic BN modeling the relationship between six reactor systems and components: the Direct Reactor Auxiliary Cooling System (DRACS), four Electro-Magnetic Pumps (EMP), and the scram system, one unmonitored physical state: differential pressure, 12 monitored plant parameters (pressure, coolant temperature, cold pool temperature, fuel temperature, power, reactivity, doppler broadening, radial expansion, axial expansion, coolant feedback, power to flow, and cladding thickness), and two accident states: TOP and LOF.

ALADDIN depends on the user providing a pre-built model such as the one shown in Figure 1. The user builds the model in GeNIe based on the requirement that at least some of the nodes are placed on a temporal plate to express their time-dependent nature. The example model shows that the four EMPs influence the amount of differential pressure; we assume each pump has the same amount influence. The arrows show the dependency between the nodes. Though they point from the reactor systems to the plant parameters, the influence is bi-directional. ALADDIN populates the conditional probability tables for each of the monitored parameters (which are located on the temporal plate) using calculations based on the given reactor simulation data.

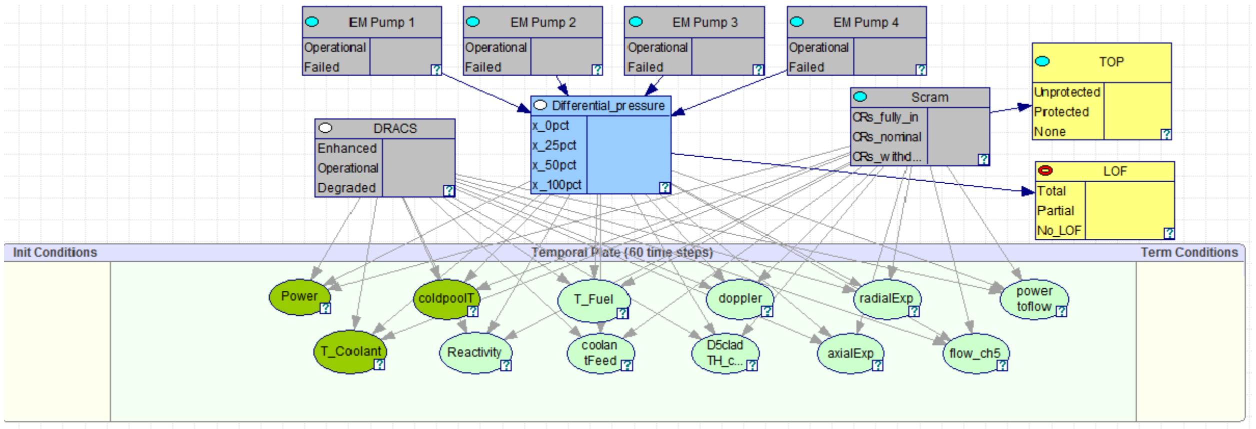


Figure 1. Structure of the Bayesian Network. The unobserved gray and blue target nodes represent reactor systems. The observed green nodes represent plant parameters located on a temporal plate since they are time dependent.

1.2 Structure of this Report

Chapter 1 outlines the motivation for the development of ALADDIN. Chapter 2 outlines the approach and methodologies. Chapter 3 is the manual for using ALADDIN. Chapter 4 contains conclusions, and outlines future work.

2 ALADDIN Approach and Methodology

2.1 Bayesian Network Basics

The output from the Discrete Dynamic Event Tree (DDET) accident simulations provides a rich data set for training supervised machine learning algorithms. The learning algorithm we chose for this task was a Dynamic Bayesian Network (DBN), a derivative of Bayesian Belief Networks (BBN). BBNs are probabilistic graphical models with random variables represented by nodes. The conditional dependencies between random variables are represented by arcs [6]. Dynamic BNs extend this by allowing the variables to change over time. We used nuclear reactor simulation data to build a DBN capable of learning the plant parameters associated with user specified accidents.

2.2 Calculation of Conditional Probabilities for Plant Parameters

The DBN must learn the conditional dependencies between the observed plant parameter variables and unobservable reactor system variables. To do this, we calculate the probability that the observed variables are in a particular state for each possible combination of the unobserved reactor target states. For example, if a DBN model contains two targets, A and B, each of which can be in two states a_1 or a_2, and b_1 or b_2, and there is one observed variable C that can be in one of two states (c_1, or c_2), then we calculate eight probability values for C (see table 1).

Once the DBN has learned these conditional probabilities, it can then be tested. During the learning phase, we define the states of the unobserved reactor systems and learn the conditional probabilities of the observed parameters; during testing we define the states of the observed variables to see the likely states of the unobserved reactor systems.

2.3 Structure of ALADDIN

The structure of the Automatic Loader of Accident Data for a Dynamic Inferencing Network, or ALADDIN, is shown in Figure 2. This illustrates the processes of ALADDIN which construct a BN model based on the data from a nuclear accident simulation. For this study we used SAS4A simulation data. In general, ALADDIN can also accept data from other nuclear simulators such as MELCOR.

After the SAS4A data is read by **DataParser**, it is cataloged in **ReactorVariables**. The data is then sent to **Discretizer** which samples the time steps for each time interval and places the samples into the value bins. The intervals and value bin parameters are defined by the output from **ConfigurationParser** (which processes user-defined configurations). Once the data is properly discretized, the **Conditionals** class uses **ConProbs** to create a conditional probability tree for each parameter. **ConProbs** also stores the values of the reactor state data into **TargetVars**. The network builder then uses the **Conditionals** class to build the Bayesian network. Once the network is built,

Table 1. Example of Conditional Probability Calculation: The calculation of the probability that observed variable C is either in state c_1 or c_2 based on the states of unobserved target variables A and B

C	B	A	Probability	Scale
c_1	b_1	a_1	$p_1 = (c_1 \mid a_1, b_1)$	
c_1	b_1	a_2	$p_2 = (c_1 \mid a_2, b_1)$	
c_1	b_2	a_1	$p_3 = (c_1 \mid a_1, b_2)$	
c_1	b_2	a_2	$p_4 = (c_1 \mid a_2, b_2)$	$\sum_{i=1}^{8} p_i = 1$
c_2	b_1	a_1	$p_5 = (c_2 \mid a_1, b_1)$	
c_2	b_1	a_2	$p_6 = (c_2 \mid a_2, b_1)$	
c_2	b_2	a_1	$p_7 = (c_2 \mid a_1, b_2)$	
c_2	b_2	a_2	$p_8 = (c_2 \mid a_2, b_2)$	

the **Pruner** class calculates the Kullbeck-Leibler divergence [7] value for each plant parameter to determine their relative information gain values (see section 3.7).

2.4 Preprocessing of Reactor Simulation Data for ALADDIN

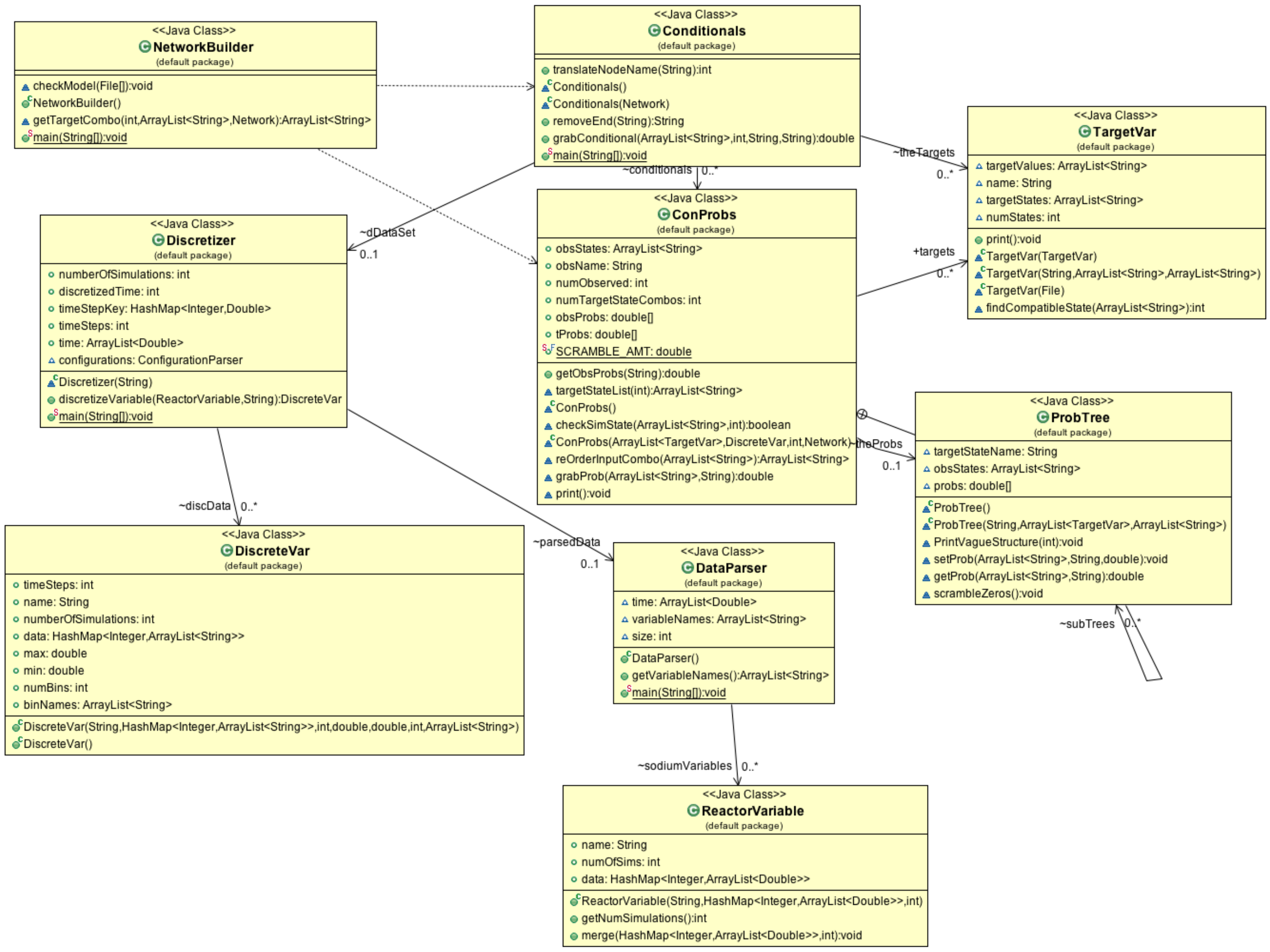
ALADDIN is designed to process reactor specifically formatted simulation data. This preprocessing entails taking reactor system and plant parameter simulation data for each accident sequence and consolidating them either into multiple files or a single file for each parameter. If the simulation data is consolidated into one file, the data from each simulation should appear in columns separated by spaces (see section 3.2).

During the simulations of accident sequences, the plant parameters can vary at every time step. The data values must be processed such that: if a parameter becomes infinite, as may occur, in SAS4a, to reactivities in the case of a cell losing all fuel to melting or all sodium to boiling, the value is replaced with zero. In similar cases, the temperatures of fuel or coolant may suddenly become zero. When this occurs, typically at the sodium boiling or fuel melting points, it is replaced with its last non-zero value.

Since the reactor systems are not dynamic in the DBN model, the state for each system for every simulation is input to the DBN from ALADDIN at the time of model construction.

2.5 Discretization

ALADDIN discretizes the inputted simulation data in two ways: by time, and by value.



<<Java Class>>
NetworkBuilder
(default package)
checkModel(File[]):void
NetworkBuilder()
getTargetCombo(int,ArrayList<String>,Network):ArrayList<String>
main(String[]):void
<<Java Class>>
Conditionals
(default package)
translateNodeName(String):int
Conditionals()
Conditionals(Network)
removeEnd(String):String
grabConditional(ArrayList<String>,int,String,String):double
main(String[]):void
~theTargets
0..*
<<Java Class>>
TargetVar
(default package)
targetValues: ArrayList<String>
name: String
targetStates: ArrayList<String>
numStates: int
print():void
TargetVar(TargetVar)
TargetVar(String,ArrayList<String>,ArrayList<String>)
TargetVar(File)
findCompatibleState(ArrayList<String>):int
~conditionals
0..*
~dDataSet
0..1
<<Java Class>>
Discretizer
(default package)
numberOfSimulations: int
discretizedTime: int
timeStepKey: HashMap<Integer,Double>
timeSteps: int
time: ArrayList<Double>
configurations: ConfigurationParser
Discretizer(String)
discretizeVariable(ReactorVariable,String):DiscreteVar
main(String[]):void
<<Java Class>>
ConProbs
(default package)
obsStates: ArrayList<String>
obsName: String
numObserved: int
numTargetStateCombos: int
obsProbs: double[]
tProbs: double[]
SCRAMBLE_AMT: double
getObsProbs(String):double
targetStateList(int):ArrayList<String>
ConProbs()
checkSimState(ArrayList<String>,int):boolean
ConProbs(ArrayList<TargetVar>,DiscreteVar,int,Network)
reOrderInputCombo(ArrayList<String>):ArrayList<String>
grabProb(ArrayList<String>,String):double
print():void
+targets
0..*
~theProbs
0..1
<<Java Class>>
ProbTree
(default package)
targetStateName: String
obsStates: ArrayList<String>
probs: double[]
ProbTree()
ProbTree(String,ArrayList<TargetVar>,ArrayList<String>)
PrintVagueStructure(int):void
setProb(ArrayList<String>,String,double):void
getProb(ArrayList<String>,String):double
scrambleZeros():void
~subTrees
0..*
~discData
0..*
<<Java Class>>
DiscreteVar
(default package)
timeSteps: int
name: String
numberOfSimulations: int
data: HashMap<Integer,ArrayList<String>>
max: double
min: double
numBins: int
binNames: ArrayList<String>
DiscreteVar(String,HashMap<Integer,ArrayList<String>>,int,double,double,int,ArrayList<String>)
DiscreteVar()
~parsedData
0..1
<<Java Class>>
DataParser
(default package)
time: ArrayList<Double>
variableNames: ArrayList<String>
size: int
DataParser()
getVariableNames():ArrayList<String>
main(String[]):void
~sodiumVariables
0..*
<<Java Class>>
ReactorVariable
(default package)
name: String
numOfSims: int
data: HashMap<Integer,ArrayList<Double>>
ReactorVariable(String,HashMap<Integer,ArrayList<Double>>,int)
getNumSimulations():int
merge(HashMap<Integer,ArrayList<Double>>,int):void

Figure 2. ALADDIN Class Diagram

2.5.1 By Time

The power plant simulators typically provide thousands of time steps for each hour of simulation time. If the BN in GeNIe were to contain the information of every time step, the time it would take to run GenIE's diagnostics tools would be untenable. Furthermore, with proper discretization, the amount of information fidelity loss is not significant enough to lessen the effectiveness of these tools. ALADDIN allows its operator to define a configuration for the discretization of each variable. The user can define the time intervals for each variable and the number of data samples which each time interval should contain. The system will draw a sample at even numbers of time steps based on the user's input. For example, suppose the user defines a time interval from 0 to 60 seconds, and the data in that interval contains time steps at increments of 0.1 seconds for a total of

600 time steps, if the user requests 60 time steps for this interval, then the system divides the total number of time steps in the interval by the number of requested time steps which defines a δ value which, in this case, would be 10. The system will take the value of the data at values 10 seconds, 20 seconds, 30 seconds, and so on until it reaches the end of the interval at 600 time steps. If the user requests a number of time steps that does not evenly divide the interval the number of time steps will slightly differ from the amount requested.

2.5.2 By Value

It has been shown that the discretization of data values is necessary when dealing with real valued input data due to the low probability of any particular value occurring[8]. The system's decision making capabilities will be less generalizable when given alternate data sets because it will have to taylor its decision boundaries to the characteristics of the particular data set. This is known as over-fitting.

The initial prototype of the system implemented the n-ary discretization method. The n-ary binning approach calculates the maximum and minimum value for each variable's range and distributes the values into n bins of equal width. The range of the parameter is calculated and then divided into 3 equal partitions. The partition containing the minimum value up to the first partition is labeled bin 0, the middle values are in bin 1, and the largest values up to the maximum value are in bin 2.

As a slight variant on the n-ary method, the system allows for the user to define the boundaries of each variable's bins as a real value or as percentages of the variable's range.

3 ALADDIN User Manual

ALADDIN is designed to automatically integrate nuclear accident simulation data (in this case SAS4A data) into a GeNIe DBN model which must be pre-built by the user. The user is required to build a DBN in GeNIe which specifies the dependency relationships between the nodes. ALADDIN reads the pre-built model, expands the dynamic nodes contained by the temporal plate (see 3), calculates the conditional probability table for each node at each discretized time step, calculates the Kullbeck-Leibler divergence value for each observed variable (see section 3.7), and writes a new Bayesian Network Model based on its calculations. Four types of files are required to run:

1. A DBN model (a file with a ".xdsl" extension)
2. SAS4A data files
3. Target files
4. Configuration files

3.1 Prebuilt GeNIe DBN model

ALADDIN depends on the user providing a pre-built model such as the one shown in Figure 3. The user builds the model in GeNIe based on the requirements that all nodes must be named to match their corresponding data files, and that time-dependent nodes are placed on a temporal plate (an unrolled network). An XDSL file must be generated by the user building a Bayesian Network in GeNIe. ALADDIN depends on a valid model being placed in the folder called "models".

3.2 SAS4A Data Files

The system depends on valid simulation data files being placed in the data folder. These files must have a ".dat" extension. The data from each simulation should appear in columns separated by spaces. The names of the files must be identical to the names of the nodes in the XDSL file. For example, if a node is named "Power", the corresponding data file must be named "Power.dat". The name is case sensitive.

It is possible to have a particular variable's data in more than one file. This system will automatically merge the additional data provided that each file is appended with an underscore and a sequence number. For example, if simulations $1 - 80$ for the Power variable are in one file, and simulations $81 - 100$ are in another file, then these two files can be placed in the data directory with the names "Power_00.dat", and "Power_01.dat". The system can handle any number of sequenced files for each variable.

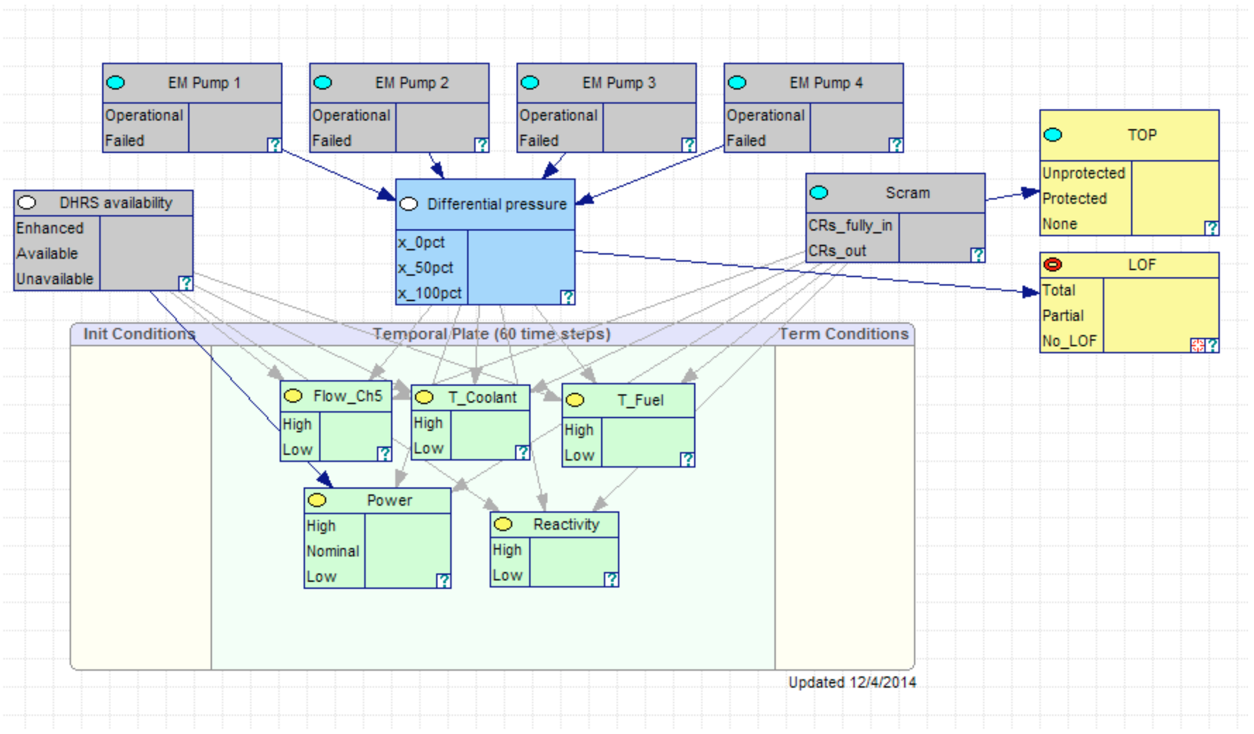


Figure 3. Example of a Bayesian Network Built With Genie.

3.3 Configuration Files

ALADDIN depends on two configuration files which must be placed the folder labeled "configurations". These configuration files must be named "bin_configurations.dat" and "time_configurations.dat".

1. **The Bin Configuration file**
   The file named "bin_configurations.dat" must list each variable in the model, the desired number of bins for that variable, the preferred discretization method, and the values for each bin. At present there are two options for the discretization method: "n-ary", and "percent". The n-ary option will discretize the variable as described in section 2.5. The option "percent" will construct bins which have boundaries defined as percentage of the range of the variable.

2. **The Time Configuration File**
   The file named "time_configurations.dat" defines intervals for the time periods in the simulations and requests the number of time steps that should be drawn from each interval.

Table 2 illustrates the user requesting that axialExp be divided into equal-sized bins by the n-ary method. In this case, the values given for bin maximums are not used and are not required. For coldpoolT, the user is requesting 3 bins. The first bin will contain the lowest 25% of the range,

Table 2. Example Bin Configuration File

Variable	Number of Bins	Method	bin0_max	bin1_max	bin2_max
axialExp	3	n-ary	33	66	100
coldpoolT	3	percent	25	50	100
coolantFeed	3	percent	33	66	100

Table 3. Example Time Step Configuration File

Interval	T_start	T_end	Steps
0	0.0	360.0	50
1	360.0	1000.0	10
2	1000.0	3600.0	1
3	3600.0	36000.0	0
4	36000.0	100000.0	2

the second bin will contain the values contained in 25% to 50% of the range, and the third bin will contain the upper half of the range.

Table 3 illustrates the user requesting 5 intervals with 50 time steps in interval 0, 10 time steps in interval 1, one time step in interval 2, 0 time steps in interval 3, and two timesteps in interval 4. The user may request as many intervals as desired. The system will draw data samples evenly across the interval but may draw more or less samples than requested depending on the amount of time steps found in each interval.

3.4 Reactor System Target Files

The target files define the states of the reactor systems for each simulation. The name of the target must be the same as the name of the target variable whose values it is describing, followed by an underscore, a number, and .dat. This means that the target files for DRACS should be named DRACS_00.dat, DRACS_01.dat, etc. Just as with the data files, the number after the target name in the file name indicates the data batch that the target file corresponds to. Thus, DRACS_00.dat would contain values of the DRACS target variable on the first set of runs, while DRACS_05.dat would be the target variable values for the 6th data set of runs. Everything is case sensitive: the target files must be named with this case sensitivity in mind: the states must be set to the correct cases depending on the input model. Thus dracs_01.dat would not work as a name for the DRACS target file. Additionally this case sensitivity also applies the state names inside each of the files.

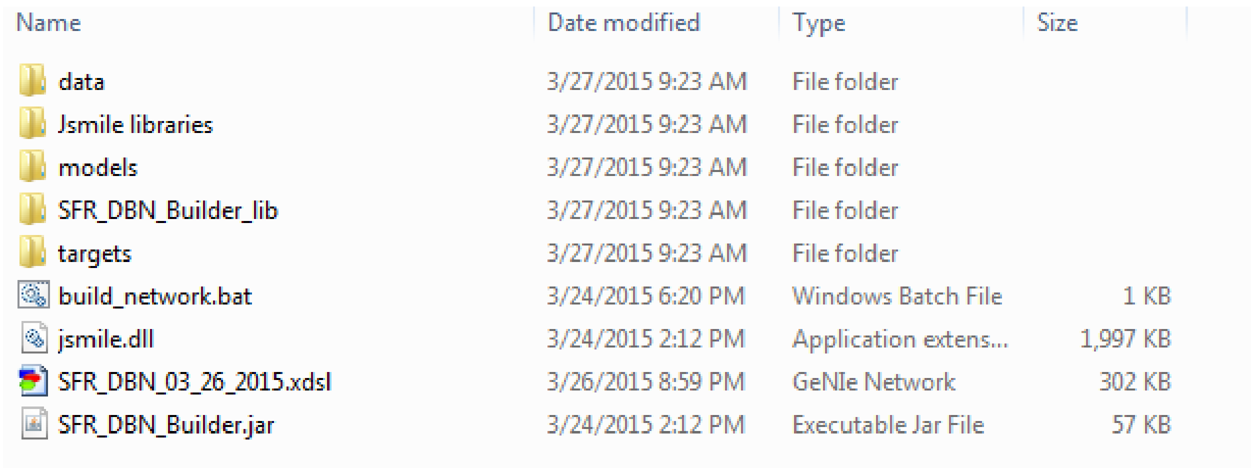
The first line of the target file contains the token "States:" followed by all the possible states the target node can take, as seen in table 4. The first line indicates that DRACS has 3 states, available, unavailable, and enhanced. The lines that follow record the state of the target variable for each of the simulation runs with a run number (starting at 0) followed by the state of the target variable during that simulation run.

Table 4. Example of Target File for DRACS

```
States: Available Unavailable Enhanced
0 Available
1 Available
2 Unavailable
3 Available
4 Enhanced
5 Available
```

3.5 ALADDIN File System

Figure 4 shows the file system of ALADDIN. The **data** folder is where all plant parameter simulation data is placed. **Jsmile libraries** contains SMILE library files. The **models** directory is where the user's pre-built GeNIe files are placed. **SFR_DBN_Builder_lib** contains the library files for ALADDIN. The **targets** directory is where the target data files are placed. **SFR_DBN_Builder.jar** is the executable file of ALADDIN which is executed by running the **build_network.bat** file. **jsmile.dll** is a windows system configuration file for running the jsmile library. **SFR_DBN_03_26_2015.xdsl** is an example of the GeNIe models generated by ALADDIN. ALADDIN generates the model with the file name appended with the current date.



Name	Date modified	Type	Size
data	3/27/2015 9:23 AM	File folder	
Jsmile libraries	3/27/2015 9:23 AM	File folder	
models	3/27/2015 9:23 AM	File folder	
SFR_DBN_Builder_lib	3/27/2015 9:23 AM	File folder	
targets	3/27/2015 9:23 AM	File folder	
build_network.bat	3/24/2015 6:20 PM	Windows Batch File	1 KB
jsmile.dll	3/24/2015 2:12 PM	Application extens...	1,997 KB
SFR_DBN_03_26_2015.xdsl	3/26/2015 8:59 PM	GeNIe Network	302 KB
SFR_DBN_Builder.jar	3/24/2015 2:12 PM	Executable Jar File	57 KB

Figure 4. The ALADDIN file system.

3.6 Executing The System In Windows

Once the data and target files and network model are in place, the system is executed by running the file called "build_network.bat". A terminal window will open and the program will execute. Upon successful completion, a file called “SFR_DBN_<Month_Day_Year_Time>.xdsl” will appear in the SFR_DBN directory. The file is appended with the time and date of the program’s execution. This xdsl file can now be opened in GeNIe for inspection.

3.7 Interpreting The Results

During the system’s execution, it will output its Kullbeck-Leibler (KL) divergence [9] calculations for each one of the plants parameter variables. The KL divergence subroutine calculates the cost associated with removing an arc or arcs from the model. Higher values signify higher costs: if the value is very low, this means that there was little or no cost associated with removing an arc.

When the program executes it outputs a KL file for every observed plant parameter variable in the network. These files contain columns corresponding to the number of target nodes in the DBN. The first column contains the cost in bits for removing the arcs between all the target variables and the observed variable. After the first column, the succeeding columns contain the KL divergence value calculated by removing the arc between the observed variable and one of each of the target variables. Each column is annotated with a header signifying the arc that was removed.

4 Conclusion

The development of ALADDIN is a step forward in Sandia's ongoing efforts to develop SMART Procedures for operator support during severe accidents in nuclear power plants. The architecture of SMART procedures includes dynamic PRA simulations of nuclear power plants to generate data for the calculation of conditional probabilities for Bayesian Networks. The graphical inferencing program GeNIe provides diagnostic tools which can be used to determine the likely state of the plant at any point during the accident sequence.

ALADDIN's role in the SMART procedure framework is to process the output of the plant simulation data and populate the Bayesian Network. The DDET based simulations typically output thousands of possible outcomes for nuclear reactor accidents. ALADDIN discretizes the simulation data, calculates conditional probability tables for each of the plants observed parameters based on the states of reactor systems, and constructs a Bayesian Network for use with GeNIe. Additionally, ALADDIN uses the Kullbeck-Leibler divergence equation to calculate the relative diagnostic value of each reactor parameter. The prototype of ALADDIN outlined in this report used SAS4A simulation data to create a diagnostic model for two accident types with twelve reactor parameters in a sodium fast reactor design. ALADDIN successfully processed the approximately 7000 accident progressions (a total of approximately 2800 megabytes of simulation data).

Future research of this system will contain a cross validation module which will identify the system's prediction accuracy. This is a standard machine learning technique which divides the data into folds. The system is repeatedly trained and tested while rotating folds for testing. This shows the performance of the system when it is trained on one set of data and tested on another. We will also investigate alternate discretization methods including: Equal Frequency Discretization: divides sorted values into bins with each bin containing equal numbers of data points, Entropy Minimization Discretization: compares the points between each successive pair of sorted values to identify those which maximize information gain, and Iterative Discretization: forms a set of intervals and then asjusts them interactively depending on a maximization criteria. The next interation of ALADDIN will also support operation in the Linux and Mac operating systems.

ALADDIN represents a major advance in the SMART procedures framework as it automates the process of parsing and transferring reactor simulation data into GeNIE BNs. This allows for scalability in model size. Whereas the previous hand-quantified model took one week to build, ALADDIN builds BN models in a matter of minutes. Additionally, ALADDIN uses information theory to calculate the relative pertinence of the plant parameters. ALADDIN contributes significantly to the tools within the SMART procedures framework for supporting operators during beyond design basis accidents.

References

[1] K. M. Groth, M. R. Denman, J. N. Cardoni, and T. A. Wheeler, "Proof of principle framework for developing dynamic risk-informed severe accident management guidelines," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2013-8324, September 2013.

[2] R. M. Summers, R. K. Cole Jr, E. A. Boucheron, M. K. Carmel, S. E. Dingman, and J. E. Kelly, "MELCOR 1.8.0: A computer code for nuclear reactor severe accident source term and risk assessment analyses," Sandia National Laboratories, Albuquerque, NM, Tech. Rep., 1991.

[3] J. E. Cahalan, A. M. Tentner, and E. E. Morris, "Advanced LMR safety analysis capabilities in the SASSYS-1 and SAS4A computer codes," Argonne National Lab, Chicago, IL, Tech. Rep., 1994.

[4] M. J. Druzdzel, "SMILE: Structural modeling, inference, and learning engine and GeNIe: a development environment for graphical decision-theoretic models," in *Proceedings of American Association for Artificial Intelligence (AAAI-99)*, 1999, pp. 902–903.

[5] K. M. Groth, M. R. Denman, J. N. Cardoni, and T. A. Wheeler, ""Smart Procedures": Using dynamic PRA to develop dynamic, context-specific severe accident management guidelines (SAMGs)," in *Proceedings of the International Conference on Probabilistic Safety Assessment and Management (PSAM 12)*, Honolulu, HI, 22–27 June 2014.

[6] A. Darwiche, "A differential approach to inference in Bayesian networks," *Journal of the ACM (JACM)*, vol. 50, no. 3, pp. 280–305, 2003.

[7] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, pp. 79–86, 1951.

[8] Y. Yang and G. I. Webb, "A comparative study of discretization methods for naive-Bayes classifiers," in *Proceedings of PKAW*, vol. 2002, 2002.

[9] R. G. Cowell, "Conditions under which conditional independence and scoring methods lead to identical selection of Bayesian network models," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2001, pp. 91–97.

DISTRIBUTION:

1	MS	0748	Katrina Groth, 6231
1	MS	0748	Matthew Denman, 6231
1	MS	0748	Michael Darling, 6231
1	MS	0748	Timothy Wheeler, 6231
1	MS	0748	Mitch McCrory, 6231
1	MS	0748	Randy Gauntt, 6232
1	MS	0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories