INL/EXT-21-63255

Light Water Reactor Sustainability Program

Development and Release of the Methods and Tools for Risk-Informed Asset Management

July 2021

U.S. Department of Energy

Office of Nuclear Energy

INL/EXT-21-63255

Development and Release of the Methods and Tools for Risk-Informed Asset Management

C. Wang, D. Mandelli, M. Abdo, A. Alfonsi, J. Cogliati, P. Talbot,
S. Lawrence, C. Smith (INL)
D. Morton (Northwestern University)
I. Popova (Texas State University)
S. Hess (Jensen Hughes)
C. Pope, J. Miller, S. Ercanbrack (Idaho State University)

July 2021

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy

EXECUTIVE SUMMARY

Under the Risk-Informed Systems Analysis Pathway, the Risk-Informed Asset Management (RIAM) project has been tasked with the development of effective and efficient analytical methods and tools to support risk-informed decisions related to plant equipment reliability and asset management programs. Under the term asset, we include not only structures, systems, and components (SSCs) and plant budget but also personnel and time. In addition, management is looked at in a broad sense, which ranges from the evaluation of future scenarios (i.e., what-if analysis) to the search for optimal plant conditions. Thus, plant resources management is translated here as the analysis and optimization of SSC lifecycle (e.g., SSC maintenance or replacement schedule), personnel job schedule (e.g., maintenance task scheduling), and maintenance/capital funds allocation (e.g., project prioritization). This is done in a risk-informed context where risk is broadly defined to include plant safety, operational reliability, and economics. The final outcome of the RIAM project is the development and release of a set of computational tools that can be deployed in the nuclear industry to support management of plant resources both over the short term and throughout the remainder of the facility lifecycle (including periods of license extension). While the development of these tools has been reported during FY-19 and FY-20 reports, this report presents the actual release of these tools for use by operating nuclear power plants. The RIAM toolkit consists of several tools that are designed for specific use cases ranging from reliability modeling to economic analysis. In this document, these tools are described in detail. For each of them, we present their designed use cases, their main functionalities, and lastly their development in terms of software quality assurance.

CONTENTS

FIGURES

TABLES

ACRONYMS

AM	Asset Management
BWR	Boiling Water Reactor
CIVET	Continuous Integration, Verification, Enhancement, and Testing
DOE	(United States) Department of Energy
ER	Equipment Reliability
GRA	Generation Risk Assessment
INL	Idaho National Laboratory
INPO	Institute of Nuclear Power Operation
IRR	Internal Rate of Return
LWR	Light Water Reactor
LWRS	Light Water Reactor Sustainability
MSPI	Mitigating Systems Performance Index
NEUP	Nuclear Energy University Program
NPP	Nuclear Power Plant
NPV	Net Present Value
NRC	(United States) Nuclear Regulatory Commission
PI	Profitability Index
PRA	Probabilistic Risk Assessment
PWR	Pressurized Water Reactor
RAVEN	Risk Analysis Virtual ENvironment
RIAM	Risk-Informed Asset Management
RISA	Risk-Informed Systems Analysis
RTM	Requirements Traceability Matrix
SQA	Software Quality Assurance
SR^2ML	Safety, Risk, Reliability Model Library
SSCs	Structures, Systems, And Components
TEAL	Tool for Economic AnaLysis
VERT	Versatile Economic Risk Tool

1. INTRODUCTION

Industry equipment reliability (ER) and asset management (AM) programs are essential elements that help ensure the safe and economical operation of nuclear power plants (NPPs). The effectiveness of these programs is addressed in several industry developed and regulatory programs. For example, all U.S. NPPs have implemented the ER process defined in INPO AP-913 "Equipment Reliability Process Description" [1]. Additionally, the performance of plant structures, systems, and components (SSCs) is monitored within a regulatory context in the Maintenance Rule 10 CFR 50.65 [2] and the Mitigating Systems Performance Index programs (MSPI) [3]. However, these programs have proven to be labor intensive and expensive. There is an opportunity to significantly enhance the collection, analysis, and use of this information to provide more cost-effective plant operation while not impacting plant operation or safety. Additionally, due to the combination of technological advancements made over the past several decades and cost pressures being experienced by the operating fleet of NPPs, there exists an acute industry need to leverage this advanced technology to reduce costs and improve operational effectiveness.

The goal of the Risk-Informed Systems Analysis[1] (RISA) [4] pathway under the Light Water Reactor Sustainability (LWRS) [5] program[2] is to provide effective and efficient analytical methods and tools to support risk-informed decisions for the ER and AM programs at NPPs. This is accomplished by creating a direct bridge, see Figure 1, between component health and lifecycle data and decision-making (e.g., maintenance scheduling, project prioritization and actuation planning). This framework supports decisions of a NPP system engineer regarding maintenance activity scheduling and component aging management. This is performed in a risk-informed context, where risk broadly includes both plant safety, operational reliability, and economics. The vision is to reduce plant operation and maintenance costs by automating this decision process, by evaluating the reliability and economic impact of different maintenance strategies (i.e., evaluate what-if scenarios), and by identifying the optimal maintenance posture that maximizes system reliability and minimizes operational costs (i.e., maximize the maintenance value given resource constraints).

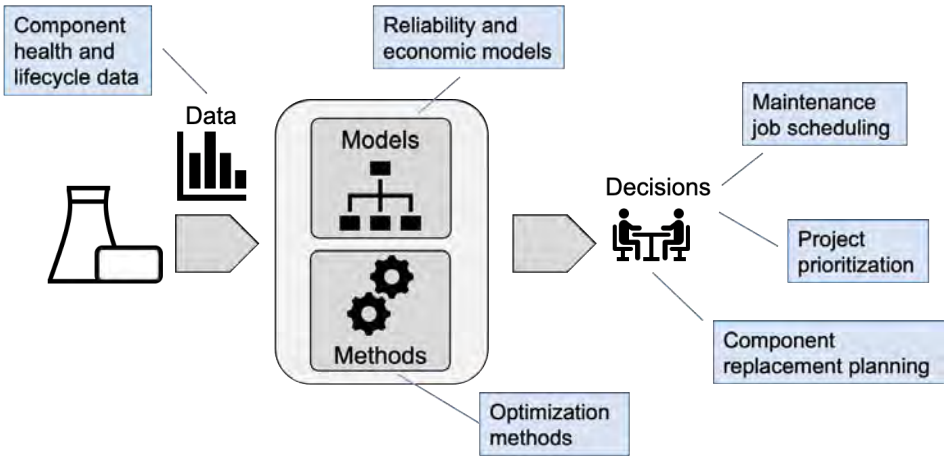


Figure 1. Graphical representation of the enterprise risk analysis framework from data to decisions.

[1] https://lwrs.inl.gov/SitePages/Risk-Informed%20Systems%20Analysis.aspx

[2] https://lwrs.inl.gov/SitePages/Home.aspx

The outcome of this project is an enterprise risk analysis framework, which can be deployed across the nuclear industry. This framework combines data analytics tools to analyze ER data with risk-informed methods designed to support system engineer decisions (e.g., maintenance and replacement schedules, optimal maintenance posture for plant systems) in a customizable workflow. A challenge is that the structure of this workflow strongly depends on the decision that needs to be made, the type of data available, and the constraints that need to be considered. Current methods are designed to provide specific answers to specific problems; however, these methods might prove to be inadequate when problem settings even slightly change (e.g., different types of constraints, additional dependencies between system reliability and economics). This project tackled this challenge by designing the framework in a flexible and modular fashion such that the user can assemble and customize their own workflow that integrates SSC economic lifecycle models (e.g., maintenance and replacement costs), system reliability models, and optimization methods (see Figure 2).

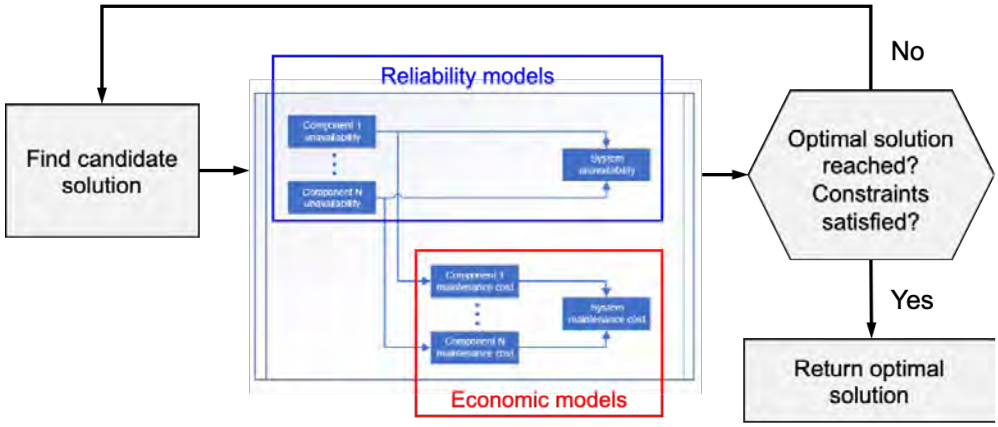


Figure 2. Example of a workflow that integrates economic and reliability models to obtain an optimal solution (e.g., optimal component replacement schedule).

In January 2021, the first step toward the development of this framework was taken when two software packages, which are an integral part of the enterprise risk analysis framework, were released with an open-source license: (1) LOGOS and (2) Safety, Risk, Reliability Model Library (SR^2ML). SR^2ML is a software package that contains a set of reliability models designed to integrate ER data (e.g., aging, testing, maintenance) and perform system level reliability calculations. LOGOS is a software package that contains a set of discrete optimization algorithms that can be employed to effectively manage plant assets and optimize schedules for plant operations. These software packages are plug-ins that can interface with the Idaho National Laboratory (INL) developed Risk Analysis and Virtual ENvironment (RAVEN) code to propagate data uncertainties (e.g., component remaining useful life) and perform data analysis and model optimization (e.g., via genetic algorithm heuristics). In this respect, SR^2ML is designed to propagate ER knowledge to the system and plant level in order to identify and evaluate the components that are most critical to the system and plant health. LOGOS uses this information to prioritize and schedule plant operations (e.g., maintenance, testing, replacement) based on budget, reliability, and resource constraints.

In this report, the software elements that are part of the enterprise risk analysis framework are described in detail regarding their designed use cases, their main functionalities, and lastly their development in terms of software quality assurance. The report focuses on LOGOS and SR^2ML and their interactions with RAVEN. In addition, two other tools that have been developed to support economic analyses related to system and plant health related to ER are described: the Tool for Economic AnaLysis (TEAL) and the

Versatile Economic Risk Tool (VERT). TEAL contains models to perform economic analyses, while VERT contains plant generation risk assessment (GRA) models for pressurized-water reactors (PWRs) and boiling-water reactors (BWRs).

This report has been structured to provide an initial overview of these tools as part of the risk analytics toolkit in Section 2 by identifying their target application and their use cases. Sections 3 through 7 describe each tool in more detail. The report provides information about the structure of the repository of each tool, the set of available models and methods, their associated use cases, and the provided documentation. Lastly, Section 8 provides a detailed summary about the software quality assurance (SQA) process that was followed and the software development procedure.

While the release of RAVEN and TEAL has been documented in other Department of Energy (DOE) reports (see [6] and [7]), this report provides also the user manuals of LOGOS, SR^{2}ML, and VERT for completeness in the three appendixes (i.e., APPENDIX A – SR^{2}ML USER MANUAL, APPENDIX B – LOGOS USER MANUAL, and APPENDIX C – VERT USER MANUAL).

The specific application of these tools are described in detail in previous LWRS reports [8],[9],[10],[11].

2. RISK ANALYTICS TOOLKIT

As mentioned in Section 1, the RIAM risk analytics toolkit is composed by several software elements, with each of them designed for a specific and well-defined application. Figure 3 shows the composition of the RIAM risk analytics toolkit by employing a tree structure.

The development of LOGOS, SR^{2}ML, and VERT has been performed mainly under the RISA-RIAM project, while the development of RAVEN and TEAL has been shared among other LWRS projects (e.g., Plant Fuel Reload Optimization project) and several other DOE projects (e.g., the Integrated Energy Systems[3] [12], and Nuclear Energy University Program (NEUP) research projects [13]). Figure 3 also indicates a target application for each software element.

Table 1 expands Figure 3 by providing a more detailed overview of each software package in terms of designed application and the set of targeted use cases. The main rationale behind the development of several tools rather than one is the simplified tool development process (e.g., smaller burden on regression testing), the easier management of tool software architecture (e.g., definition of model dependencies), the easier integration with other DOE projects, and the ability for operating NPPs to choose the relevant tool (or combination of tools) that best support the needs of a particular plant application.

An important note to highlight is that these software tools are categorized and constructed as RAVEN plug-ins. This architecture guarantees the possibility to combine elements of different tools together to perform complex analyses when desired. As we have indicated also in [10] and [11], the analysis that needs to be performed might require either a single software element (i.e., standalone configuration), or a combination of multiple software elements. An example of analysis based on a standalone configuration is the selection of the optimal set of projects out of a pool of candidate projects that maximize overall value (i.e., the net present value [NPV]) to support investment decision-making at the plant or fleet level. In the simple case where projects are independent and good estimates of project costs and benefits are available, the knapsack model in LOGOS could be directly employed (see [9]). In this case, the user provides estimates of the NPV of each project and evaluates the outcomes provided by LOGOS to provide information to executive decision-makers. However, for more complex situations, the NPV of a project

[3] https://ies.inl.gov/SitePages/Home.aspx

might not be directly available at hand (e.g., due to dependencies with other projects or decisions) but needs to be computed at each iteration through the evaluation and decision-making process. This type of problem could be solved by employing the economic models in TEAL, which can be directly linked to the knapsack model in LOGOS by employing the RAVEN ensemble model capability.

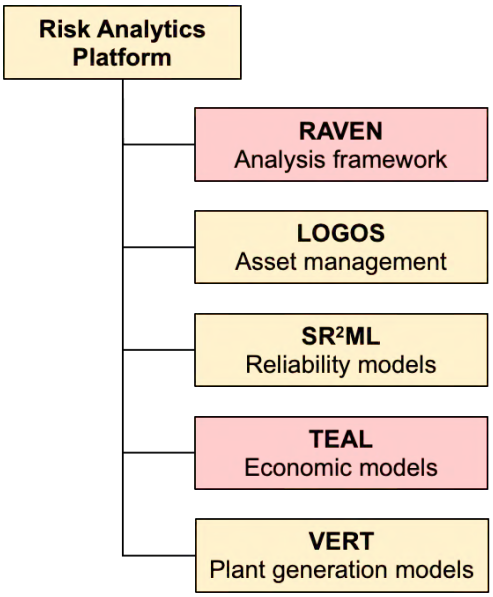


Figure 3 RIAM risk analytics toolkit. Elements in yellow were developed within the RIAM project while the development of the elements in red was shared among other DOE programs.

Table 1. Summary of the element composing the RIAM risk analytics platform.

Name	Application	Use Cases
LOGOS	Plant resource optimization	Project prioritization Project scheduling Task scheduling
SR^2ML	Reliability modeling	Component reliability models Margin-based solver ER data analysis methods
RAVEN	Data analysis computational framework	Model optimization Uncertainty propagation Data analysis Linking models
TEAL	Economic analysis	Economic cashflow models
VERT	Plant models	Light water reactor (LWR) GRA models LWR probabilistic risk assessment (PRA) models

Sections 3 through 7 provide more details of each software element of the RIAM risk analytics toolkit. In particular, the focus is on the repository structure, contained computational elements, provided documentation, and software license.

3. LOGOS

LOGOS (see Figure 4) contains a set of discrete optimization models that can be employed for plant resource optimization problems [16]. LOGOS integrates economic and reliability risks into a single analysis framework. More specifically, provided SSC health information (e.g., failure rate or failure probability), operation and maintenance costs, replacement costs, other costs associated with component failure, and plant budget constraints, LOGOS determines the optimal set of projects (e.g., SSC replacement or refurbishment) that maximizes profit and satisfies the provided requirements/constraints.

Figure 4. Screenshot of the LOGOS GitHub repository (https://github.com/idaholab/LOGOS).

The input data listed above can be either deterministic or stochastic in nature (i.e., they can be point forecasts or probability distribution functions). In the latter case, several scenarios are generated by sampling the provided distributions. The developed models are based on enhanced versions of the knapsack optimization problem. The basic knapsack optimization problem for capital budgeting can be defined as follows: given is an instance of a capital budgeting problem with an investment set N, which consists of n investments i with profit p_i, and cost w_i, and the available budget c. The objective is to select a subset of N such that the total profit of the selected investments is maximized and the total cost does not exceed c. More details on the formulation of the knapsack problem are provided in reference [11].

The two primary application classes are: capital budgeting optimization and task schedule optimization. The goal for the capital budgeting is to determine the optimal set of projects (i.e., project prioritization) and their schedule that maximize overall profit. The goal for the task scheduler is to determine the optimal project schedule that minimizes the overall completion time.

The developed plant models can be either deterministic or stochastic. Deterministic models are simpler since they require point value data associated to constraints such as project costs and rewards. Stochastic optimization models extend deterministic models by explicitly considering data uncertainties. These formulations can be employed as standalone models or interfaced with the INL-developed RAVEN code to propagate data uncertainties and analyze generated data (i.e., sensitivity analysis and uncertainty quantification).

The internal structure of LOGOS is represented in Figure 5, which highlights the location of the source code, provided documentation, and set of regression tests. Table 2 provides a basic summary of the LOGOS repository in terms of development language, license, and GitHub repository. LOGOS heavily relies on the PYOMO library (http://www.pyomo.org/) developed by Sandia National Laboratories and University of California, Davis. The PYOMO library provides a large set of optimization models and solver engines in Python. Table 2 also lists the number of regression tests that are currently used to monitor new developments and source code changes and updates.

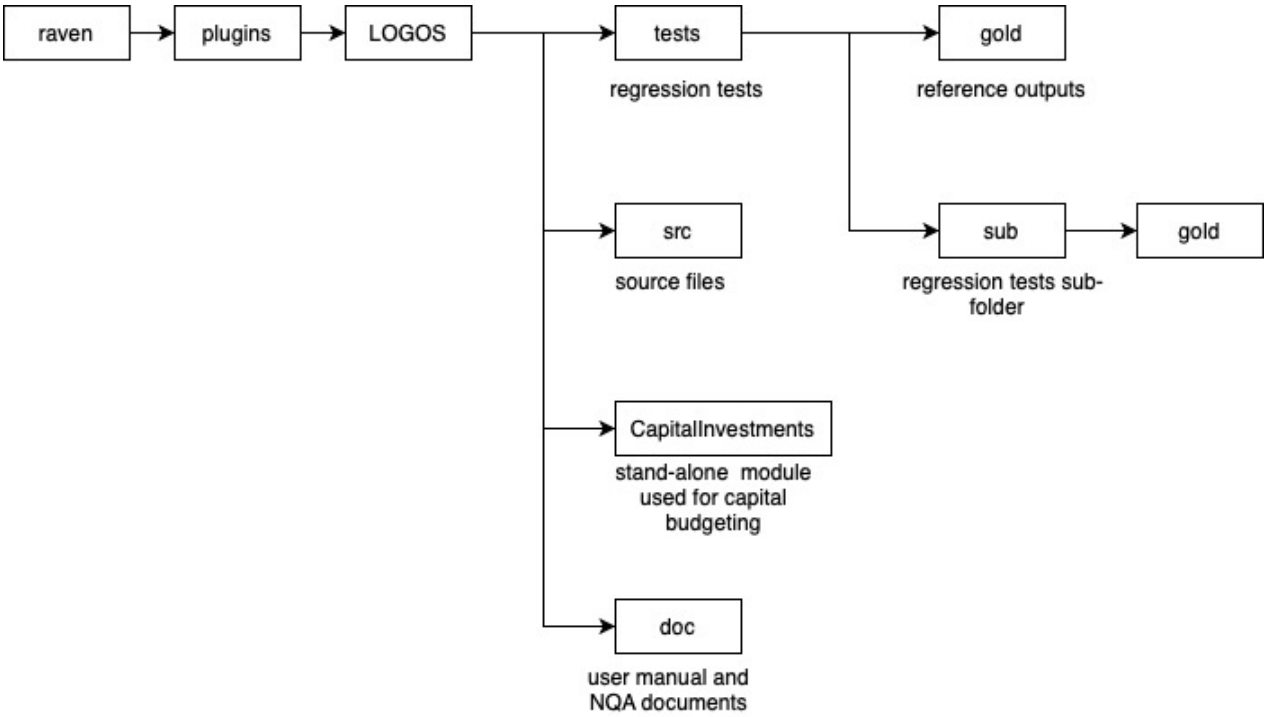


Figure 5. Folder tree of the LOGOS repository.

Table 2. Summary of LOGOS architecture.

Development language	Python
Dependencies	PYOMO[4] [14],GLPK[5] and CoinCBC[6]
Repository site	https://github.com/idaholab/logos
License	Apache-2
Number of regression tests	60
Supported operating system	macOS, Linux, Windows

Table 3 provides a more detailed overview of the actual methods and models contained in LOGOS. The first two listed in Table 3 are designed to be used in a model-based optimization setting (see Figure 2) where the user specifies the architecture of the overall system model, which might include economic (e.g., from TEAL), reliability (e.g., from SR^{2}ML), and LOGOS sub-models (e.g., project scheduling model). Then, the user would link models together using the RAVEN ensemble model capability and perform model optimization on the integrated model using optimization engines (e.g., genetic algorithms) available in RAVEN.

The rest of the methods and models listed in Table 3 are designed to mainly be used in a standalone configuration to perform data-based optimization-type analyses. If desired, these models and methods can be linked to RAVEN to propagate data uncertainties (see [9]).

Table 3. List of LOGOS models and methods and their corresponding use case.

Model/Method Class	Use Case
Simple and multiple knapsack model	Model-based project prioritization, selection, and scheduling
Scheduling model	Model-based tasks scheduling
Simple and multiple knapsack optimization model	Data-based project prioritization, selection, and scheduling using PYOMO
Distributionally robust simple and multiple knapsack optimization model	Data-based project prioritization, selection, and scheduling using distributionally robust formalism using PYOMO
Risk-based simple and multiple knapsack optimization model	Data-based project prioritization, selection, and scheduling using conditional value-at-risk formalism using PYOMO

[4] http://www.pyomo.org/

[5] https://www.gnu.org/software/glpk/

[6] https://projects.coin-or.org/Cbc

4. SR^{2}ML

The SR^{2}ML (see Figure 6) is a software package that contains a set of reliability models designed to interface with RAVEN. These models can be employed to perform both static and dynamic system risk analysis and determine the risk importance of specific elements of the considered system. Two classes of reliability models are included:

- The first class includes all classical reliability models (fault trees, event trees, Markov models, and reliability block diagrams), which have been extended to deal not only with Boolean logic values but also time-dependent values. These models have been designed to mainly perform simulation-based reliability analyses where time needs to be explicitly considered.
- The second class includes several reliability models designed to incorporate the effect of component aging and maintenance on component unavailability. These models have been designed to support decision-making regarding maintenance posture at the plant and system level (e.g., optimal maintenance operations that best balance cost and reliability).

Models included in these two classes are designed to be included in a RAVEN ensemble model to perform time-dependent system reliability analysis (e.g., dynamic analysis). Similarly, these models can be interfaced with system analysis codes within RAVEN to determine the failure time of systems and evaluate accident progression (static analysis).

The reliability analyses that can be performed in SR^{2}ML are not only classical (i.e., probability based) but also more innovative using a margin-based language. Classical reliability modeling is performed by coupling existing PRA quantification codes (e.g., SAPHIRE) with SR^{2}ML models. The main goal of margin-based reliability analyses is to better integrate ER data in order to assess system health. Traditional reliability models (e.g., fault trees) are still employed, but a different calculation engine is used. In this innovative calculation engine, the risk metric is expressed in terms of "margin to failure" rather than "probability of failure". The most important point of this method is how ER data generated by a component under different maintenance approaches (e.g., corrective or condition-based) can be used directly to measure component margin. Note that this method is not designed to be an alternative approach to current PRA methods, but instead a complementary approach designed for different kinds of decisions. This approach aims to assist decision-making related to plant resources and AM activities, such as work scheduling and project prioritization. Note that additional details on the functionality and structure of SR^{2}ML have been provided in the previous report [10].

In addition, during FY-21, the development of a set of machine-learning methods designed to analyze ER data using numeric and natural language processing algorithms has started. These methods are designed to be integrated directly to plant monitoring and diagnostic centers in order to analyze current data and compare it with historic data. The targeted applications range from SSC diagnostic/prognostic to current risk-informed applications such as the surveillance frequency control (SFCP) program.

The internal structure of SR^{2}ML is represented in Figure 7. The figure indicates the key modules of the SR^{2}ML application as well as available documentation (of particular note being the user manual, software requirements specification, and software design specification which can be accessed via the GitHub https://github.com/idaholab/SR2ML). Also of note is the suite of regression tests used for configuration control to ensure appropriate functionality of the software.

Figure 6. Screenshot of the SR^{2}ML GitHub repository (https://github.com/idaholab/SR2ML).

Table 4 provides a more detailed overview of the actual methods and models contained in SR^{2}ML. For each method/model the type of analysis that can be performed is specified (probability or margin-based) and the specific use case of reference.

Table 5 provides a basic summary of SR^{2}ML repository in terms of development language, license, and GitHub repository.

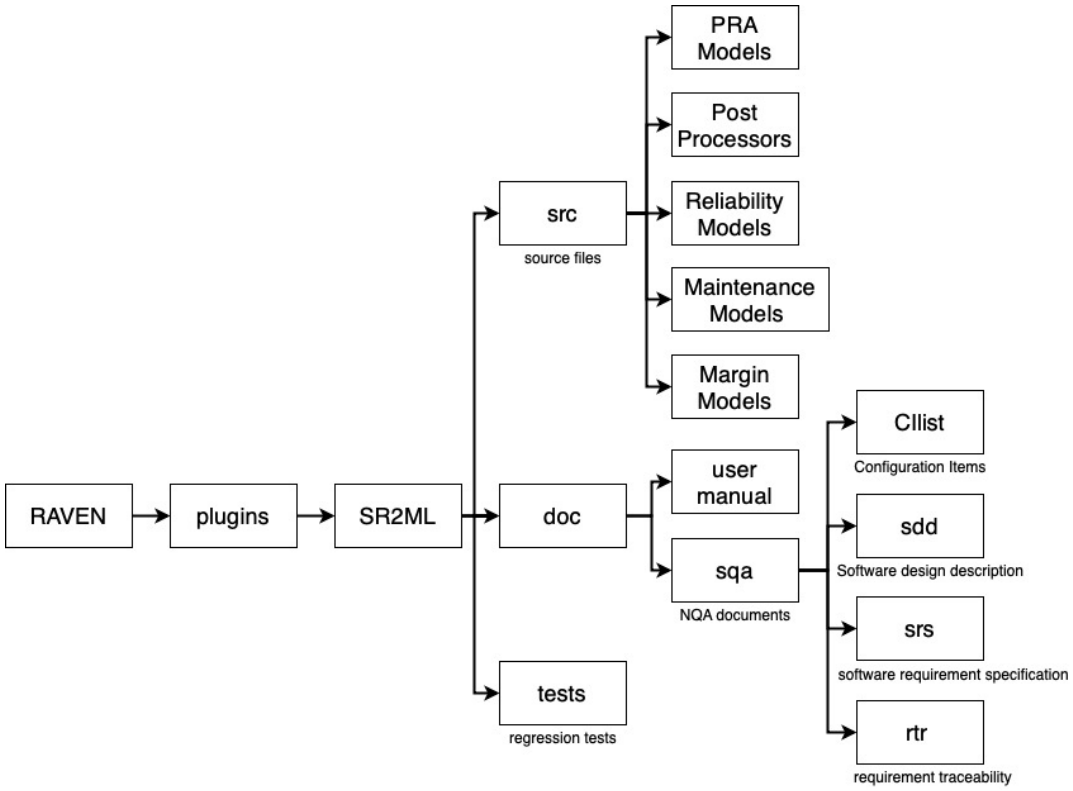


Figure 7. Folder tree of the SR^{2}ML repository.

Table 4. List of SR^{2}ML models and methods and their corresponding use case.

Model/Method Class	Analysis Type	Use Case
Maintenance models	Probability based	Models designed to calculate availability and unavailability for component under preventive maintenance conditions
Reliability models	Probability based	Models designed to calculate component reliability for different aging profiles
Margin models	Margin-based	Models designed to calculate component margin for both point and time series values
Fault tree, event tree, graph, and reliability block diagram models	Simulation-based	Models designed to perform simulation-based reliability analysis using classical reliability models
Minimal cut set solver	Probability based	Model designed to perform probability calculation provided set of minimal cut sets
Margin-based reliability solver	Margin-based	Code designed to perform margin-based reliability calculations
ER data analysis methods	—	Library of machine-learning methods designed to analyze ER data using numeric and natural language processing algorithms

Table 5. Summary of SR^{2}ML architecture.

Development language	`Python`
Dependencies	RAVEN
Repository site	https://github.com/idaholab/SR2ML
License	Apache-2
Number of regression tests	67
Supported operating system	macOS, Linux, Windows

5. RAVEN

RAVEN [6] is a flexible and multipurpose modeling and simulation platform designed to perform uncertainty quantifications, regression analyses, data analyses, and model optimization analyses (see Figure 8). Depending on the tasks to be accomplished and on the probabilistic characterization of the problem, RAVEN perturbs the response of the system by altering its parameters through Monte Carlo, Latin hypercube, and other reliability surface search [15] sampling methods.

The data generated by the sampling process are analyzed using classical and more advanced data mining approaches. RAVEN also manages the parallel dispatching (i.e., both on desktop, workstation, and large high-performance computing machines) of the software representing the physical model. RAVEN heavily relies on artificial intelligence algorithms to construct surrogate models of complex physical systems to perform uncertainty quantification, reliability analyses (e.g., limit state surface), and parametric studies.

Figure 8. Graphical representation of RAVEN computational capabilities.

Table 6 provides a summary of the subset of methods and models contained in RAVEN that are being used within the RIAM project. More specifically, RAVEN is actively used to:

- Link models (e.g., contained in SR^{2}ML and/or LOGOS) together using, for example, the ensemble and logical model
- Sample the constructed models (e.g., for data uncertainty propagation and model optimization) using RAVEN samplers, optimizers (e.g., genetic algorithms), and the "RAVEN running RAVEN" capability (e.g., mix of uncertainty propagation and model optimization)
- Postprocess the generated data using classical statistical and data mining methods.

Lastly, Table 7 provides a basic summary of RAVEN repository in terms of development language, license, and GitHub repository.

Table 6. List of RAVEN models and methods and their corresponding use case.

Model/Method/Capability	Use Case
Samplers (e.g., Monte Carlo, Latin hypercube, grid)	Methods designed to perform uncertainty quantification by sampling model input variables according to specific strategy
Pareto frontier [17]	Method designed to determine the Pareto frontier from a multidimensional dataset
Optimizers (e.g., gradient based, genetic algorithms [18], simulated annealing)	Methods designed to determine the global minima and maxima of the output variable of a model by smartly changing input variable values
Ensemble model	Capability to link models together in a linear data workflow
Logical model	Capability to choose which model to run (out of two) based on a set of logical conditions
Code interface	Capability to link external codes (e.g., PRA, economic or thermohydraulic codes) and treat them as RAVEN models
RAVEN running RAVEN	Capability to perform a two-level analysis where the first RAVEN is considered as a model (slave mode) while the second RAVEN (master mode) performs any sampling strategy (e.g., sampling or optimization)
Basic statistics	Postprocessor designed to analyze large datasets with the goal of determining the statistical moments (e.g., mean and variance) and perform sensitivity analysis
Data mining	Postprocessor designed to perform clustering of large datasets to evaluate similarities and patterns from the generated data

Table 7. Summary of RAVEN architecture.

Development language	Python, C++
Dependencies	Check https://github.com/idaholab/raven/blob/devel/dependencies.xml for a complete list
Repository site	https://github.com/idaholab/raven
License	Apache-2
Number of regression tests	745
Supported operating system	macOS, Linux, Windows

6. VERT

The VERT is a GRA software application focused on simplifying, while improving, evaluation of NPP electricity generation (see Figure 9 and Figure 10). VERT uses GRA methods to examine how component reliability and degradation impact the ability for an NPP to generate electricity (see Figure 11 for a tree representation of the VERT repository structure). Whereas PRA uses fault trees to focus on scenarios impacting plant nuclear safety, GRA uses fault trees to focus on scenarios impacting plant electricity generation, including temporary plant power output derating (see Table 8 for a summary of VERT targeted use cases). Combining GRA with a cost model allows for direct economic analyses.

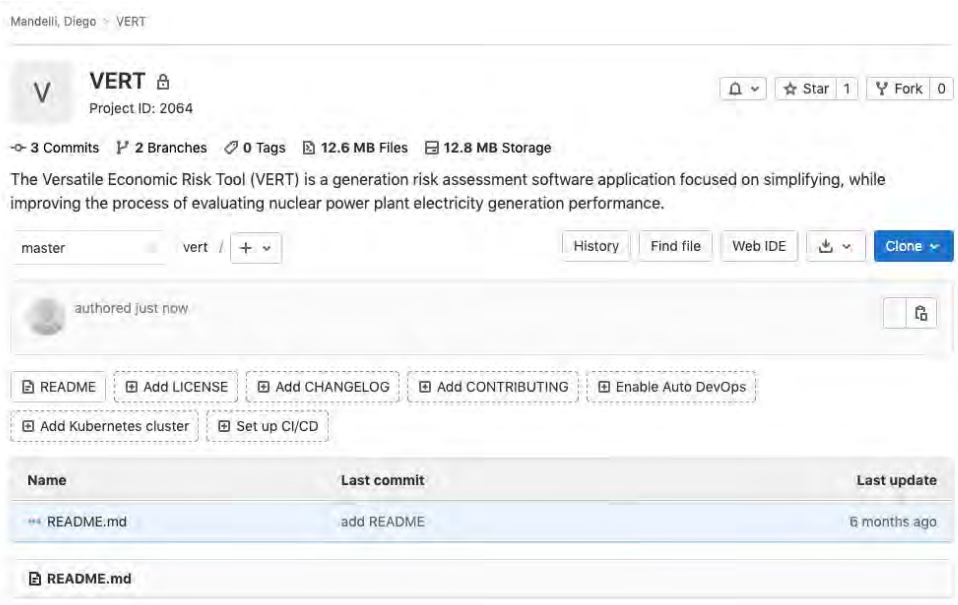


Figure 9. Screenshot of the VERT GitLab repository (https://hpcgitlab.inl.gov/mandd/vert).

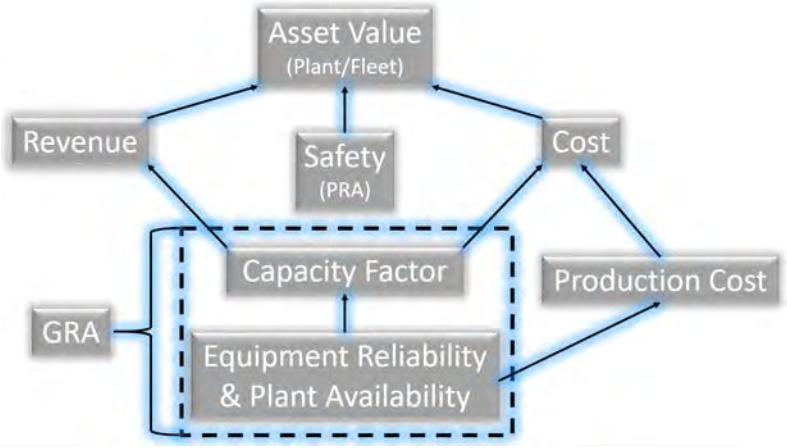


Figure 10. GRA role in power plant asset management.

VERT couples the INL-developed SAPHIRE fault tree evaluation software [19] and RAVEN, which provides a framework for parametric variability along with automated results production and analysis (see Figure 12). Using component reliability data and degradation models, VERT can identify SSCs that significantly contribute to plant electricity generation loss over time (see Figure 13). The identified high-risk equipment can then be targeted for reliability and degradation improvements. Similarly, VERT can be used to identify components that are not significant contributors to a risk of loss of electricity generation. This equipment may be subject to over-conservative reliability and maintenance activities which could be relaxed to obtain cost savings. Insights from VERT support the optimization of maintenance, inspection, and other electricity generation improvement strategies. Note that the functionality and structure of VERT have been provided in a previous report [10]. Lastly, Table 9 provides a basic summary of the VERT repository in terms of development language, license, and GitHub repository.

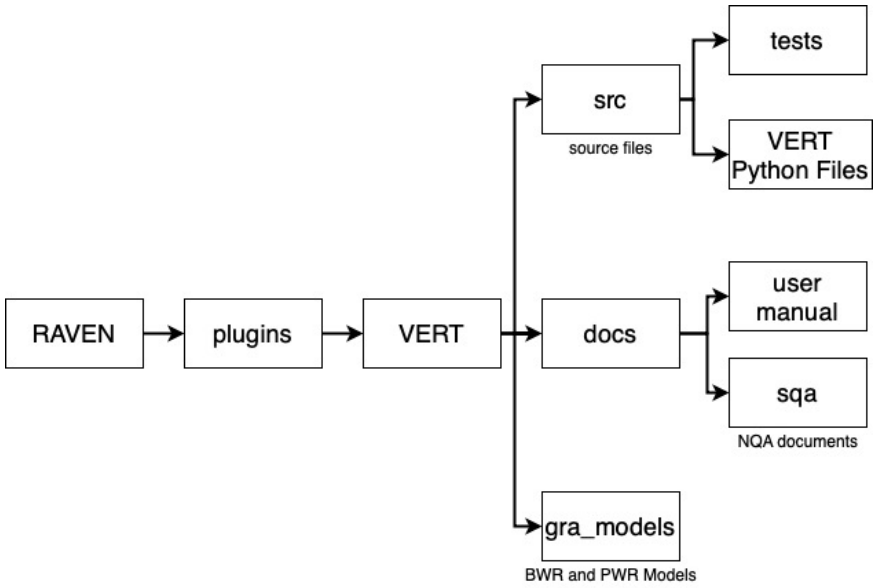


Figure 11. Folder tree of the VERT repository.

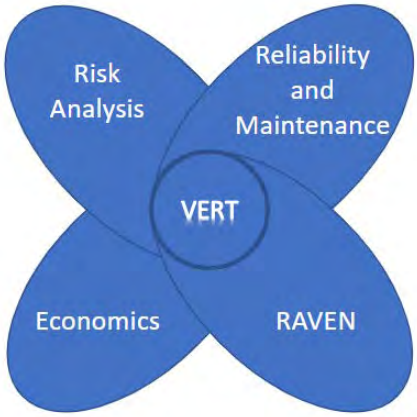


Figure 12. VERT conceptual schematic.

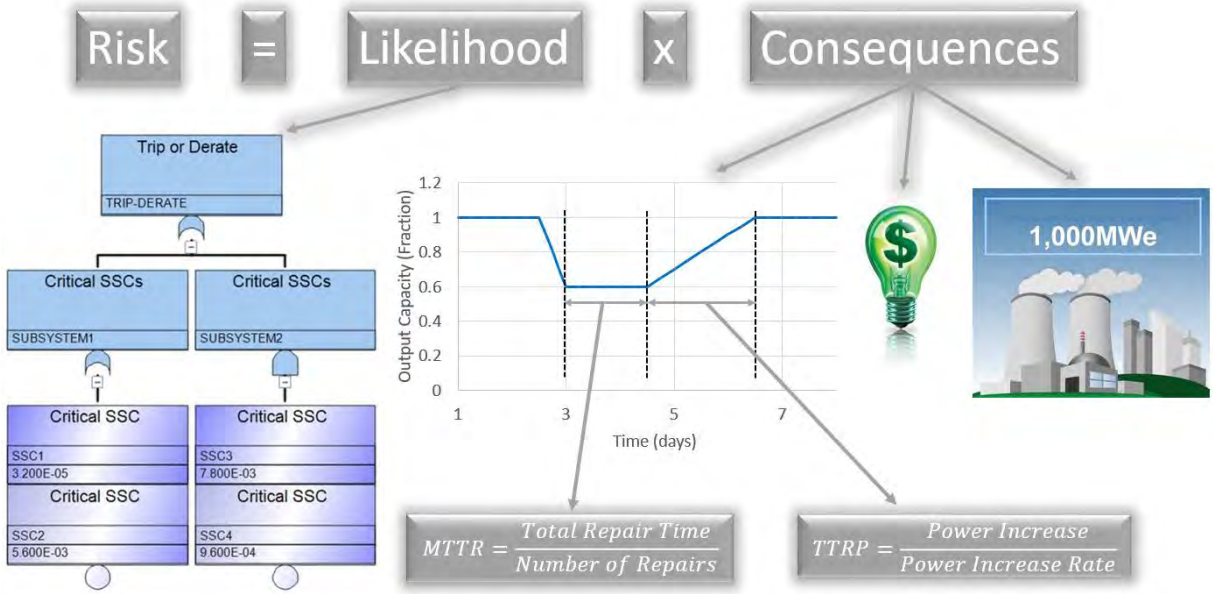


Figure 13. VERT risk methodology diagram.

Table 8. List of VERT models and methods and their corresponding use case.

Model/Method	Use Case
GUI for GRA model construction	Automated creation of PWR/BWR GRA models and interface with SAPHIRE and RAVEN
Generic LWR GRA models	Reference GRA models for generic PWR and BWR for benchmarking purposes
Generic LWR PRA models	Reference PRA models for generic PWR and BWR for benchmarking purposes

Table 9. Summary of VERT architecture.

Development language	Python
Dependencies	RAVEN, SAPHIRE
Repository site	https://hpcgitlab.inl.gov/mandd/vert
License	NDA agreement (to be released open-source in the near future)
Number of regression tests	NA
Supported operating system	Windows

7. TEAL

This package enables the capability to compute the NPV, internal rate of return (IRR), and profitability index (PI) with RAVEN. Furthermore, TEAL makes it possible to perform an NPV, IRR, or PI search. TEAL computes a multiplicative value (e.g., the production cost) so that the NPV, IRR, or PI produces a desired value (such as a desired Minimum Acceptable Rate of Return (MRR) or so called "hurdle rate" for investment decisions). The plug-in allows for a generic definition of cash flows with drivers provided by RAVEN. Furthermore, TEAL includes flexible options to deal with taxes, inflation, and discounting and offers capabilities to compute a combined cash flow for components with different component lives. Note that the primary analysis methods incorporated in TEAL have been provided in a previous report [11]. Table 10 provides a summary of TEAL targeted use cases while Table 11 provides a basic summary of the TEAL repository in terms of development language, license, and GitHub repository

Table 10. List of TEAL models and methods and their corresponding use case.

Model/Method	Use case
Cashflow	Model designed to perform cashflow calculations over the lifecycle of a project or component
Amortization	Model designed to perform amortization calculation

Table 11. Summary of TEAL architecture.

Development language:	Python
Dependencies:	RAVEN
Repository site:	https://github.com/idaholab/TEAL
License:	Apache-2

8. SOFTWARE QUALITY ASSURANCE PRACTICES

SQA is a set of necessary activities to provide adequate confidence that a software item or product conforms to the specified set of functional and technical requirements. The SQA plan presents the standardized method for capturing software requirements and the required activities to enable consistent

SQA implementation (i.e., how the requirements will be implemented, how the software will be tested, how changes to the software will be controlled, and how software deficiencies will be handled). The SQA plan presented in this report establishes the SQA program for RAVEN and RAVEN plug-ins discussed in the previous sections.

8.1 SQA Documentation

RIAM tools inherit RAVEN SQA practices and documentation from RAVEN. They fully adopt and follow the RAVEN SQA process and plan. The RAVEN SQA documents are maintained both in the RAVEN GitHub repository (doc/sqa) and INL EDMS system. RAVEN is currently classified as NQ1 Level 3 (research tool) software[7], but it will soon be moved and maintained with Level 2 (commercial grade – no safety) standard. As RAVEN plug-ins, RIAM tools are currently classified as NQ1 Level 3 software. The SQA documents of RIAM tools are also maintained both in their own GitHub repositories (doc/sqa) and the INL EDMS system.

The RAVEN SQA plan is condensed in *PLN-5552, RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan*. This plan is currently maintained in the RAVEN GitHub repository (doc/sqa/sqap), and this plan covers the following main SQA documents and sections:

- Software Quality Assurance Plan (SQA)
  This plan presents the required activities to enable consistent SQA implementation within RAVEN software and RAVEN plug-ins (i.e., RIAM tools)
- Configuration Management Plan
  This plan documents the configuration management activities, plan management, and maintenance needed to assure proper configuration of RAVEN software and RAVEN plug-ins. Specifically, it outlines the configuration identification, change controls, status accounting, evaluation, and reviews.
- Software Standards and Reviews Plan
  This plan provides software coding standards, commentary standards, and testing standards.
- Software Testing Plan
  This plan outlines the test procedures, the type of tests, test automation, and requirement tests.
- IT Asset Maintenance Plan
  This plan documents the maintenance and operation activities (i.e., backup and recovery, management and operating [M&O] planning, assessment and control) for RAVEN and RAVEN plug-ins
- Verification and Validation Plan
  This plan presents the verification and validation plan processes (i.e., pull request testing, code review, development branch testing, and master branch testing) for RAVEN and RAVEN plug-ins.

To adopt and follow the above plan, RIAM tools as RAVEN plug-ins produce the following supplementary documents as listed in Table 12.

[7] More information can be found at https://github.com/idaholab/raven/wiki/external-plugin-SQA-requirements .

Table 12. RIAM tools supplementary documents.

Document Type	Document Description
Base documents/digital entries	Safety software determination, quality level determination, and asset management entry (for INL Enterprise Architecture Entry)
Software requirements specifications	Software requirements specifications have a complete description of the features and behavior of each RIAM tool. The document includes various elements describing the minimum functionality, usability, and performance requirements of the software. It also captures how the system interfaces and operates with external applications
Requirements Traceability Matrix (RTM)	RTM is a document that relates to user requirements on the test cases for each RIAM tool to validate the completeness of the software. The parameters included in RTM is the requirement ID, the requirement description, and the test case for verification of every requirement
Configuration item list	The configuration item list aims to gather software, hardware, and documentation components of each RIAM tool's infrastructure. These are the items that define the configuration of the software
Software design description	The software design description is a document written by a software designer or architect to give other software developers an overall guide to the architecture of the software product. It has an architectural diagram with detailed information on system purpose and scope, along with the structure of code and software. It serves as a reference document outlining all parts and functionality of the RIAM tool
User manual	This document describes the capabilities and intended use of the software within specified limits

In order to retrieve the SQA documentation, navigate to the RIAM tool SQA directory located at:./ToolName/doc/sqa/ and run (in your terminal) the building bash script:

```
./make_docs.sh
```

All the following SQA documents are generated and collected in a new folder called "sqa_built_documents":

- Configuration items list
- Software design description
- Software requirements specification and traceability matrix

The dashboards of the required documentation for SR^{2}ML and LOGOS are illustrated in Figure 14 and Figure 15, respectively.

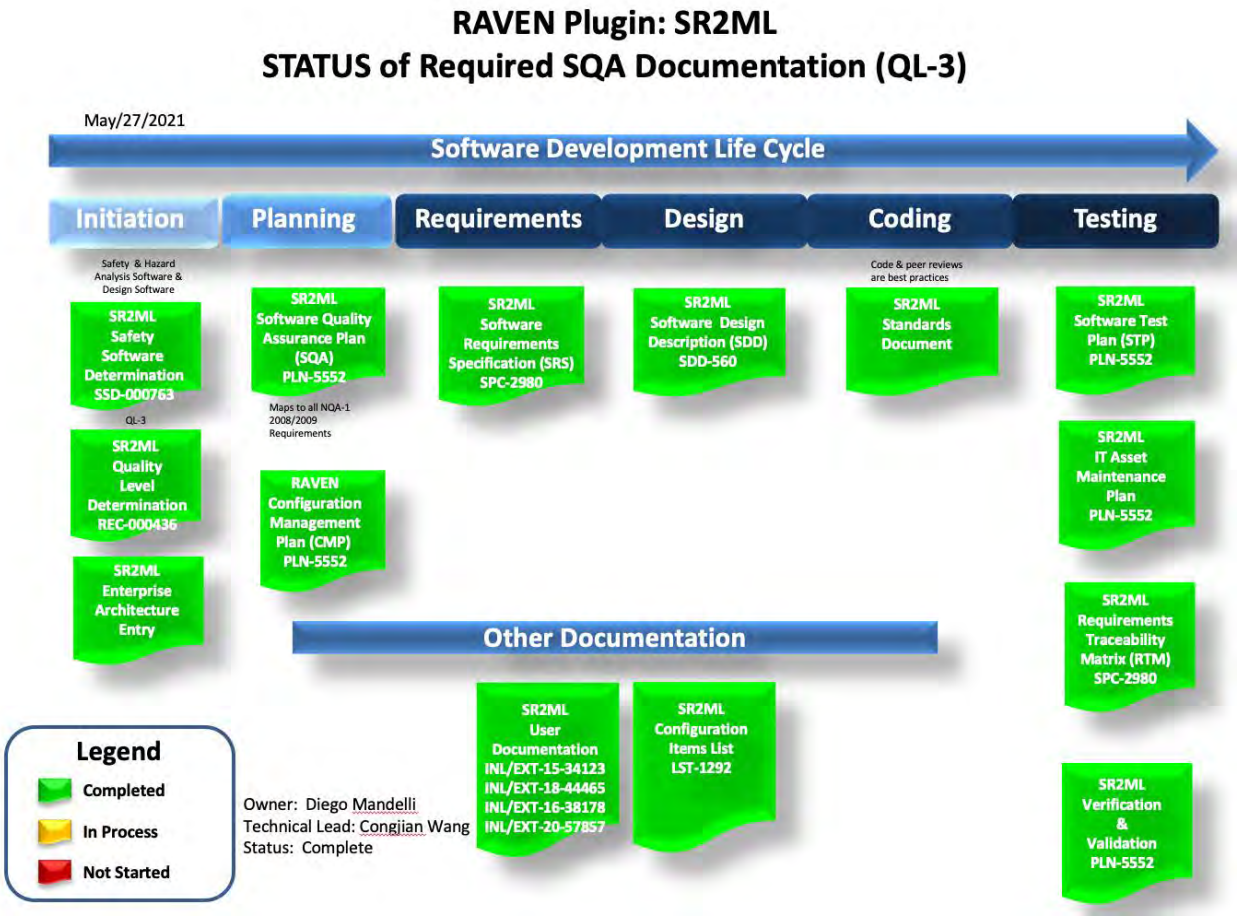


Figure 14. Dashboard of required documentation for SR^{2}ML.

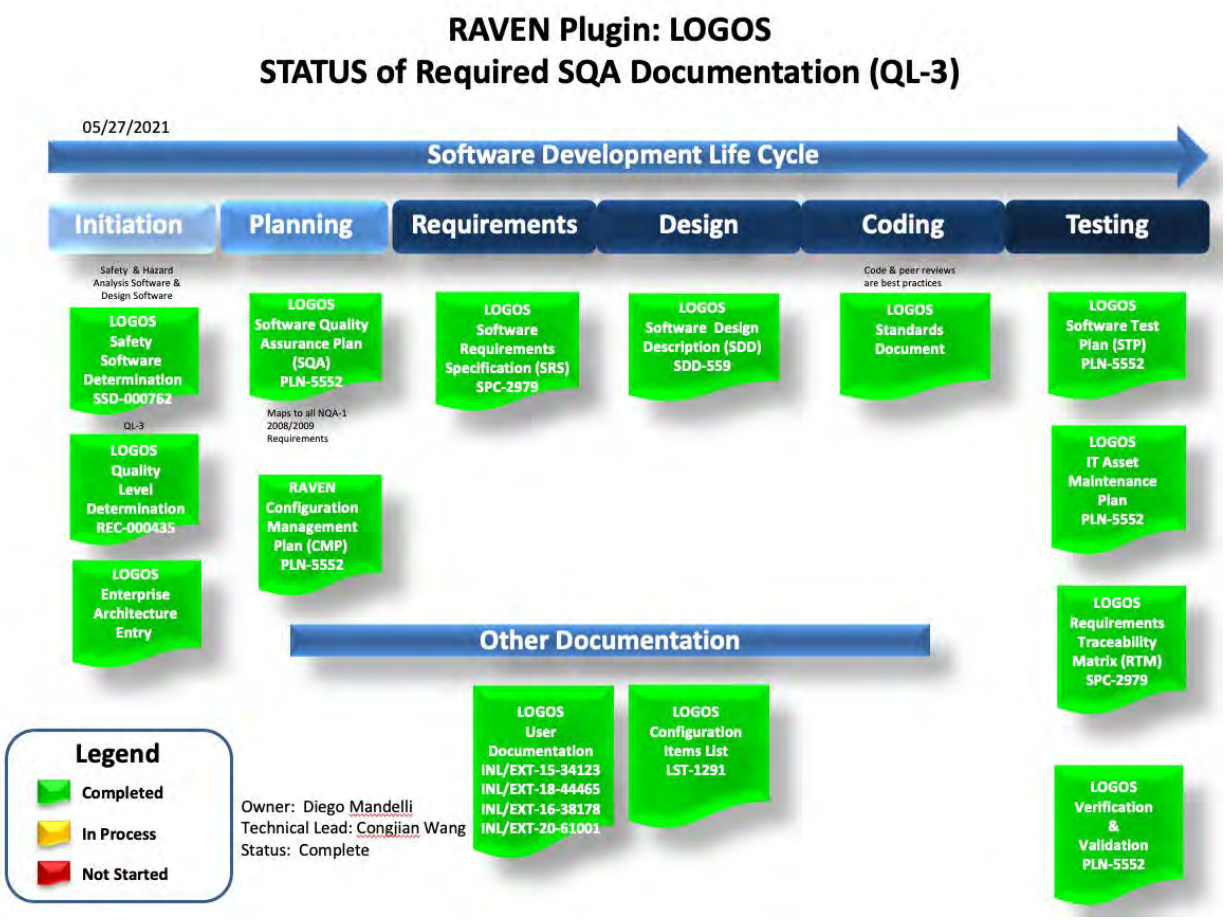


Figure 15. Dashboard of required documentation for LOGOS.

8.2 Configuration Control Process

RIAM tools also adopt and follow RAVEN configuration control process as illustrated in Figure 16 and Figure 17. The change control activities consist of requesting a change, performing an evaluation, obtaining an approval (or disapproval), providing notification to requester, and executing design, implementation, acceptance testing, and closure of changes. Changes include both error correction and enhancement. The GitHub *Issues* and *Pull requests* system and INL CIVET (Continuous Integration, Verification, Enhancement, and Testing system) are currently used to track and log the activities through the change control process.

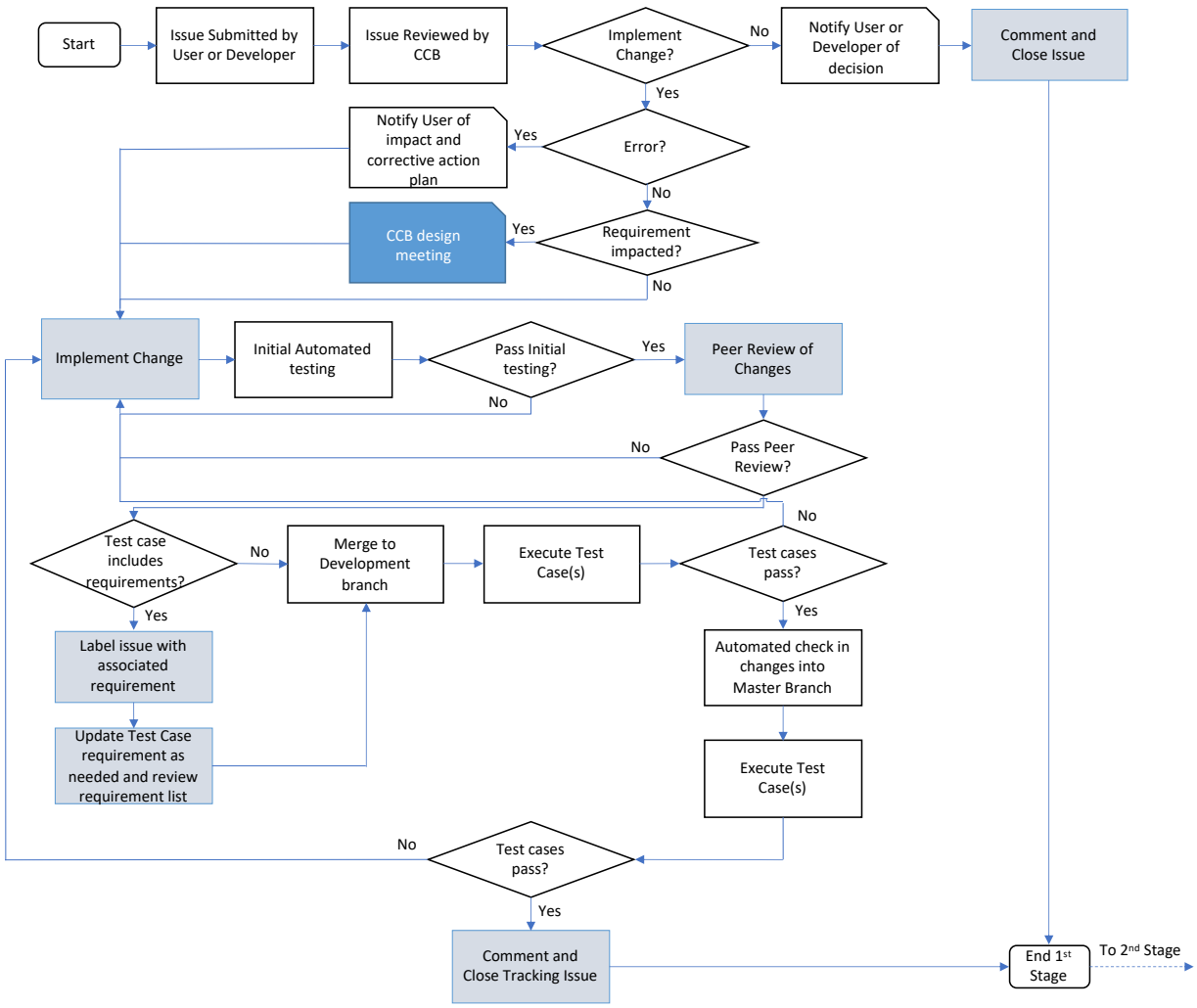


Figure 16. RAVEN and RAVEN plug-ins configuration control process (1st stage).

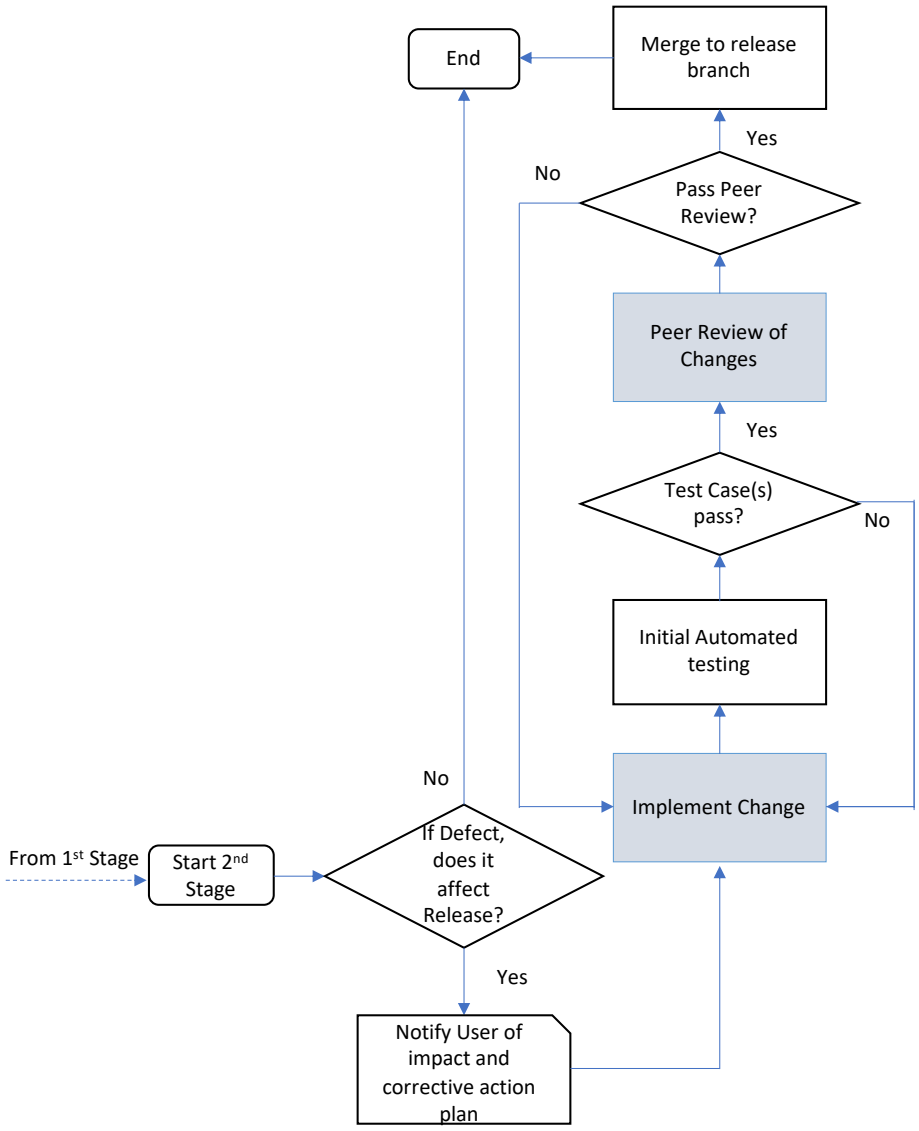


Figure 17. RAVEN and RAVEN plug-ins configuration control process (2nd stage).

8.3 Version Control and Accessibility

RIAM tools are currently maintained as open-source Git repositories hosted on GitHub (https://github.com). GitHub provides the distributed version control and source code management functionality of Git as well as several collaboration features, such as bug tracking, feature requests, task management, continuous integration, and wikis for each repository. As described in the SQA documentation, the development work is performed on development branches and must undergo review, testing, and approval before being pushed to the master branch of RIAM tools. This enforces the control of issue resolution and feature development. The GitHub Issue System is used to track and log the discussion between issue creators and RIAM tool developers and the GitHub Pull Request System is used to manage the change request and review process. The automated tests are also managed by GitHub and all test results

are logged and tracked by the INL CIVET System[8]. The following sections illustrate details of automated regression testing and "issue and pull" request system.

8.4 Regression Testing

RIAM tools inherit the same regression testing philosophy and framework used by RAVEN; that is, before any modifications to the code can be merged into the main development branch, a series of regression tests must be passed to ensure that behavior has not notably changed, or if it has changed, that the changed behavior is documented and updated as part of the code changes. These regression tests are a combination of analytic tests evaluating the performance of RIAM tools, and integration tests evaluation the ability of RIAM tools to perform as a whole to satisfy particular use cases. There are numerous regression tests for each RIAM tool. All the test configurations are available in the code repository under ToolName/tests. For LOGOS and SR^{2}ML, a list of requirement test cases are presented and described in Table 13 and Table 14, respectively.

RIAM tools are periodically tested for use on MAC, Linux, and Windows operating systems. These automated tests are managed by GitHub, and all the test results are logged and tracked by the INL CIVET System, as shown in Figure 18 as an example.

Table 13. LOGOS requirement tests.

Functional Requirements	Requirement Description	Test
L-F-1	The LOGOS plug-in shall allow support for user-defined analysis flow using RAVEN on capital budgeting problems	LOGOS/tests/MilestoneTestsSeptFY19/use_case_III/raven_capital_budgeting_III.xml
L-F-2	The LOGOS plug-in shall allow support for standalone analysis on capital budgeting problems	LOGOS/tests/CapitalInvestments/examples/use_case_3b_fy19_consistent_constraint_I_II.xml
L-DCBO-1	LOGOS shall be able to perform single knapsack deterministic capital budgeting optimization	LOGOS/tests/CapitalInvestments/skp/test_skp.xml
L-DCBO-2	LOGOS shall be able to perform multiple knapsack deterministic capital budgeting optimization	LOGOS/tests/CapitalInvestments/mkp/test_mkp.xml
L-DCBO-3	LOGOS shall be able to perform multichoice knapsack deterministic capital budgeting optimization	LOGOS/tests/CapitalInvestments/mckp/test_mckp.xml

[8] https://civet.inl.gov/

Functional Requirements	Requirement Description	Test
L-SDCBO-1	LOGOS plug-in shall be able to perform stochastic analysis of single knapsack deterministic capital budgeting optimization	LOGOS/tests/test_skp.xml
L-SDCBO-2	LOGOS plug-in shall be able to perform stochastic analysis of multiple knapsack deterministic capital budgeting optimization	LOGOS/tests/test_mkp.xml
L-SDCBO-3	LOGOS plug-in shall be able to perform stochastic analysis of multichoice knapsack deterministic capital budgeting optimization	LOGOS/tests/test_mckp.xml
L-SCBO-1	LOGOS shall be able to perform single knapsack stochastic capital budgeting optimization	LOGOS/tests/CapitalInvestments/skp/test_multi_variation.xml
L-SCBO-2	LOGOS shall be able to perform multiple knapsack stochastic capital budgeting optimization	LOGOS/tests/CapitalInvestments/mkp/test_mkp_scenarios.xml
L-SCBO-3	LOGOS shall be able to perform multichoice knapsack stochastic capital budgeting optimization	LOGOS/tests/CapitalInvestments/mckp/test_mckp_scenario.xml
L-DRO-1	LOGOS shall be able to perform single knapsack distributionally robust capital budgeting optimization	LOGOS/tests/CapitalInvestments/dro/test_droskp_scenario.xml
L-DRO-2	LOGOS shall be able to perform multiple knapsack distributionally robust capital budgeting optimization	LOGOS/tests/CapitalInvestments/dro/test_dromkp_scenarios.xml
L-DRO-3	LOGOS shall be able to perform multichoice knapsack distributionally robust capital budgeting optimization	LOGOS/tests/CapitalInvestments/dro/test_dromckp_scenario.xml
L-CVaR-1	LOGOS shall be able to perform single knapsack stochastic capital budgeting optimization based conditional value-at-risk	LOGOS/tests/CapitalInvestments/cvar/test_cvar_skp_scenario.xml
L-CVaR-2	LOGOS shall be able to perform multiple knapsack stochastic capital budgeting optimization based conditional value-at-risk	LOGOS/tests/CapitalInvestments/cvar/test_cvar_mkp_scenarios.xml
L-CVaR-3	LOGOS shall be able to perform multichoice knapsack stochastic capital budgeting optimization based conditional value-at-risk	LOGOS/tests/CapitalInvestments/cvar/test_cvar_mckp_scenario.xml

Table 14. SR^{2}ML requirement tests.

Functional Requirements	Requirement Description	Test
SR2ML-F-1	The SR^{2}ML plug-in shall allow support for user-defined instructions for controlling the execution stages of the probabilistic safety, risk and reliability analysis.	SR2ML/tests/test_ensemblePRAModel_discrete.xml
SR2ML-F-2	The SR^{2}ML plug-in shall allow for user-defined output types for simulation data.	SR2ML/tests/test_markovModel_3states.xml
SR2ML-PRA-1	The SR^{2}ML plug-in shall support importing Event Tree Boolean logic structure to construct RAVEN Event Tree ExternalModel	SR2ML/tests/test_ETmodel.xml
SR2ML-PRA-2	The SR^{2}ML plug-in shall support importing Fault Tree Boolean logic structure to construct RAVEN Fault Tree ExternalModel	SR2ML/tests/test_FTmodel.xml
SR2ML-PRA-3	The SR^{2}ML plug-in shall support importing a generic Markov chain to construct RAVEN Markov chain ExternalModel	/SR2ML/tests/test_markovModel_2states.xml
SR2ML-PRA-4	The SR^{2}ML plug- in shall support importing Reliability Block Diagram (RBD) Boolean logic structure to construct RAVEN RBD ExternalModel	SR2ML/tests/test_graphModel.xml
SR2ML-SI-1	The SR^{2}ML plug-in shall be able to be initialized via input files.	SR2ML/tests/test_ensemblePRAModel_mixed.xml

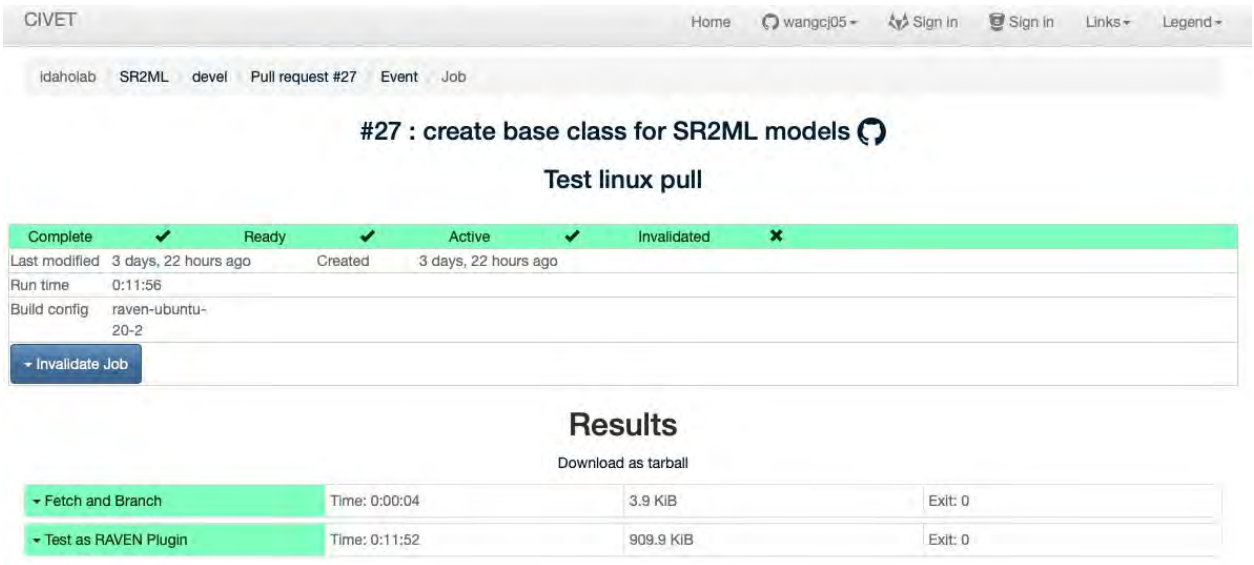
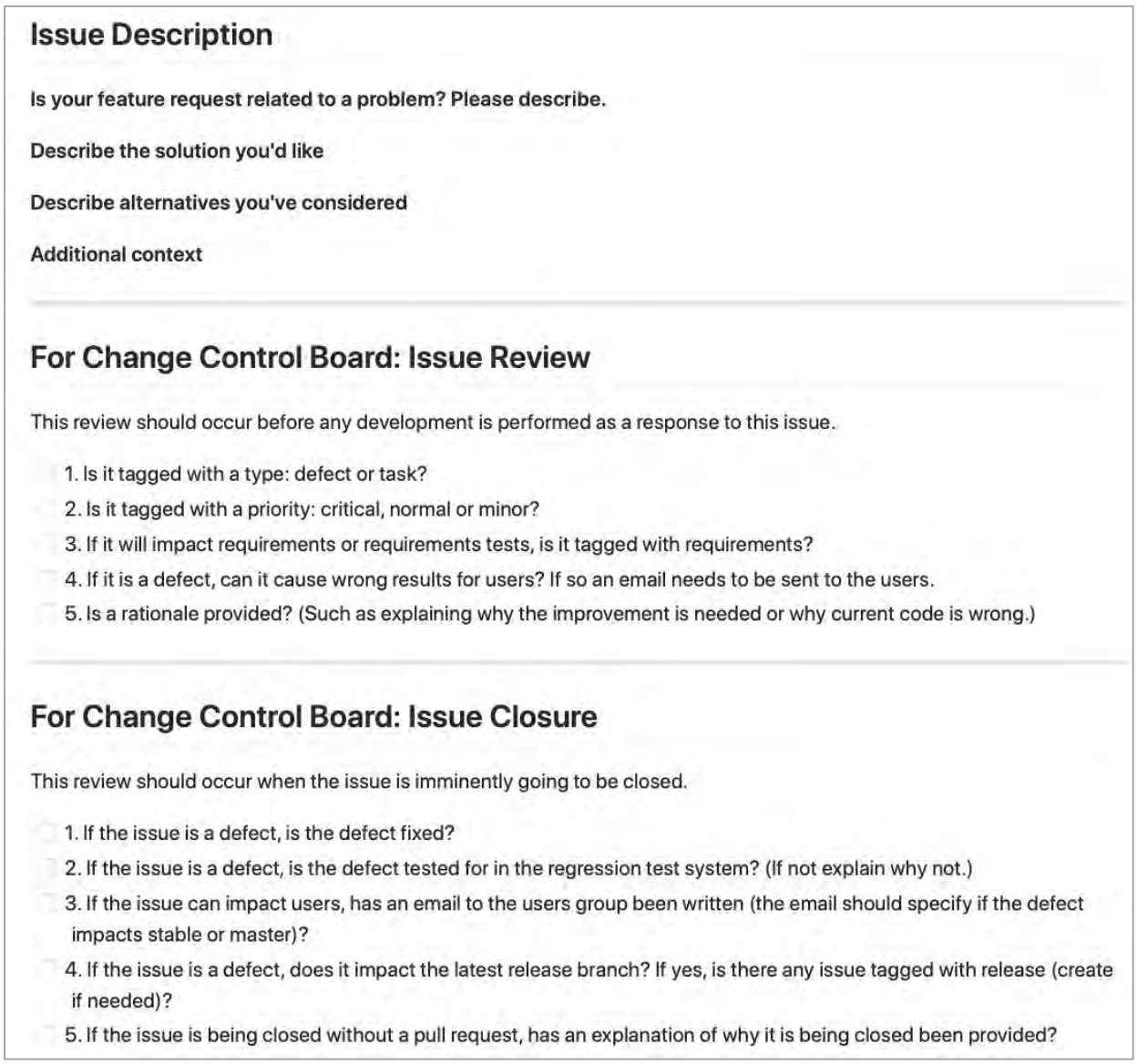


Figure 18. Regression testing process managed by CIVET (SR^{2}ML example).

8.5 Issue and Pull Request

Issues can be initiated by anyone, including offsite users, and this option is used for maintenance, new development, enhancements of the software tools, or it can be used to report program errors and problems. The GitHub Issue System is used to track and log the discussion between issue creators and RIAM tool developers. Figure 19 shows the issue template that is used for each feature request of a RIAM tool.



Issue Description

Is your feature request related to a problem? Please describe.

Describe the solution you'd like

Describe alternatives you've considered

Additional context

For Change Control Board: Issue Review

This review should occur before any development is performed as a response to this issue.

1. Is it tagged with a type: defect or task?
2. Is it tagged with a priority: critical, normal or minor?
3. If it will impact requirements or requirements tests, is it tagged with requirements?
4. If it is a defect, can it cause wrong results for users? If so an email needs to be sent to the users.
5. Is a rationale provided? (Such as explaining why the improvement is needed or why current code is wrong.)

For Change Control Board: Issue Closure

This review should occur when the issue is imminently going to be closed.

1. If the issue is a defect, is the defect fixed?
2. If the issue is a defect, is the defect tested for in the regression test system? (If not explain why not.)
3. If the issue can impact users, has an email to the users group been written (the email should specify if the defect impacts stable or master)?
4. If the issue is a defect, does it impact the latest release branch? If yes, is there any issue tagged with release (create if needed)?
5. If the issue is being closed without a pull request, has an explanation of why it is being closed been provided?

Figure 19. Issue template for RIAM tool feature request.

Pull requests can also be initiated by anyone and this option is used for maintenance, enhancement, or new capability of the software tool. Pull requests inform others about changes pushed to a branch in a repository on GitHub. Once a pull request is opened, the RIAM tool developers can discuss and view the potential changes before any changes are merged into the base branch. Figure 20 shows the pull request template that is used for RIAM tools.

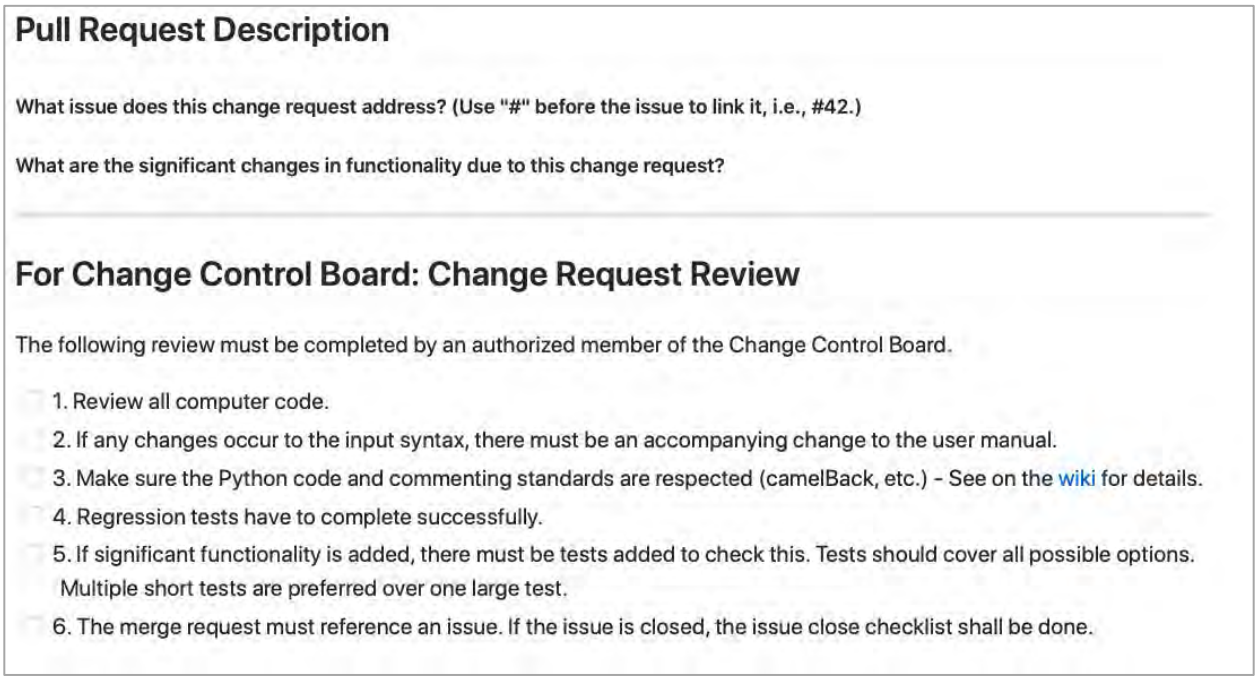


Figure 20. Pull request template for RIAM tool.

9. CONCLUSIONS

In this document, we have presented several software products designed to support AM decision-making under the RIAM project. These products have been designed for specific use cases ranging from reliability and economic modeling to model- and data-based optimization methods. Each product is available on its own Git repository, which contains the actual source code, the reference documentation (e.g., user manual and user guide), and a set of regression tests. For each product, the report presents its architecture and structure. In particular, this report has expanded the specific use cases for each method and model contained in each product. The actual application of these tools for specific types of analysis are described in [8,9,10,11]. The plan is to release all of these tools as open-source with an Apache-2 license. This will further simplify the deployment and use of these tools into existing plant health and decision-making software frameworks. Lastly, it is important to note that the development for some of the tools indicated in Figure 3 have been shared among other DOE projects within LWRS and NEUP.

REFERENCES

[1] Institute of Nuclear Power Operations (INPO). 2018. “AP-913 - Equipment Reliability Process Description, Revision 6.” Washington, DC (limited distribution).

[2] United States Nuclear Regulatory Commission (NRC). n.d. “10 CFR 50.65 Requirements for monitoring the effectiveness of maintenance at nuclear power plants.” Last modified March 24, 2021. https://www.nrc.gov/reading-rm/doc-collections/cfr/part050/part050-0065.html.

[3] United States Nuclear Regulatory Commission (NRC). 2005. “Independent Verification of the Mitigating Systems Performance Index (MSPI) Results for the Pilot Plants.” NUREG-1816, NRC, Washington, DC.

[4] H. Zhang, Y.-J. Choi, C. Smith, S. Prescott, D. Mandelli, A. Alfonsi. 2020. "Risk-Informed Systems Analysis (RISA) Pathway Technical Program Plan." INL/EXT-19-55721, Idaho National Laboratory.

[5] Idaho National Laboratory. 2020. "Integrated Program Plan." INL/EXT-11-23452, Rev.8, Idaho National Laboratory [https://lwrs.inl.gov/Technical%20Integration%20Office/LWRS_Integrated_Program_Plan.pdf#search=INL%2FEXT%2D11%2D23452]

[6] Alfonsi, A., C. Rabiti, D. Mandelli, J. Cogliati, C. Wang, P. W. Talbot, D. P. Maljovec, and C. Smith. 2021. "RAVEN Theory Manual." INL/EXT-16-38178 Rev. 4, Idaho National Laboratory [https://doi.org/10.2172/1784873].

[7] A. Alfonsi, C. Wang, A. Epiney, C. Rabiti, and USDOE Office of Nuclear Energy, "TEAL. Computer software," USDOE Office of Nuclear Energy (NE) [https://www.osti.gov//servlets/purl/1668299] (2020) Web. doi:10.11578/dc.20200929.3.

[8] Mandelli, D., Z. Ma, R. Youngblood, S. St. Germain, C. Smith, P. Talbot, S. Hess, D. Dube, C. Pope, J. Miller, M. Robbins, D. Das, M. Azarian, and J. Coble. 2019. "Plant Integral Risk-informed System Health Program." INL-EXT-19-55808, Idaho National Laboratory.

[9] Mandelli, D., C. Wang, S. St. Germain, C. Smith, D. Morton, I. Popova, and S. Hess. 2019. "Combined Data Analytics and Risk Analysis Tool for Long Term Capital SSC Refurbishment and Replacement." INL-EXT-19-55819, Idaho National Laboratory.

[10] Mandelli, D., C. Wang, J. Cogliati, C. Smith, S. Hess, R. Sugrue, C. Pope, J. Miller, S. Ercanbrack, D. Cole, and J. Yurko. 2020. "Integration of Data Analytics with Plant System Health Program." INL/EXT-20-59928, Idaho National Laboratory.

[11] Mandelli, D., C. Wang, M. Abdo, A. Alfonsi, P. Talbot, J. Cogliati, C. Smith, D. Morton, I. Popova, and S. Hess. 2020. "Development and Application of a Risk Analysis Toolkit for Plant Resources Optimization." INL-EXT-20-59942, Idaho National Laboratory.

[12] Bragg-Sitton, S. M., C. Rabiti, R. D. Boardman, J. O'Brien, T. J. Morton, S. Jong Yoon, J. Soo Yoo, K. Frick, and P. Sabharwall. 2020. "Integrated Energy Systems: 2020 Roadmap." INL-EXT-20-57708 Rev. 1, Idaho National Laboratory. https://doi.org/10.2172/1670434.

[13] Mandelli, D., C. Wang, A. Alfonsi, Z. Ma, C. Smith, C. Parisi, R. Youngblood, and T. Aldemir. 2021. "Mutual Integration of Classical and Dynamic PRA." Nuclear Technology 207 (3): 363–375. https://doi.org/10.1080/00295450.2020.1776030.

[14] Hart, W. E., J.-P. Watson, and D. L. Woodruff. 2011. "PYOMO: Modeling and Solving Mathematical Programs in Python." Mathematical Programming Computation 3 (3): 219–260.

[15] Maljovec, D., B. Wang, V. Pascucci, P.-T. Bremer, and D. Mandelli. 2013. "Adaptive sampling algorithms for probabilistic risk assessment of nuclear simulations." in *ANS PSA 2013 International Topical Meeting on Probabilistic Safety Assessment and Analysis Columbia, SC*, on CD-ROM, American Nuclear Society, LaGrange Park, IL.

[16] Hillier, F., and G. Lieberman. 2015. *Introduction to Operations Research*, 10th edition, McGraw-Hill.

[17] Mattson, C.A., and A. Messac. 2005. "Pareto Frontier Based Concept Selection Under Uncertainty, with Visualization." Optimization and Engineering 6: 85–115. https://doi.org/10.1023/B:OPTE.0000048538.35456.45.

[18] Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution programs*, 3rd edition, Springer Edition.

[19] Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE), risk and reliability assessment tool, [https://saphire.inl.gov/]

APPENDIX A – SR^{2}ML USER MANUAL

MANUAL

INL/EXT-20-57857
Revision 1
Printed July 12, 2021

SR2ML User Manual

Congjian Wang and Diego Mandelli

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

INL/EXT-20-57857
Revision 1
Printed July 12, 2021

SR2ML User Manual

Diego Mandelli
Congjian Wang

Contents

1 Introduction

SR2ML is a software package that contains a set of safety and reliability models designed to be interfaced with the INL's RAVEN code. These models can be employed to perform both static and dynamic system risk analyses and to determine the risk importance of specific elements of the considered system.

Two classes of reliability models have been developed; the first class includes all classical reliability models (fault trees, event trees, Markov models and reliability block diagrams) which have been extended to not only deal with Boolean logic values but also time-dependent values. The second class includes several component aging and maintenance models. Models of these two classes are designed to be included in a RAVEN ensemble model to perform a time-dependent system reliability analysis (dynamic analysis). Similarly, these models can be interfaced with system analysis codes to determine the failure time of systems and evaluate the accident progression (static analysis).

1.1 Acquiring and Installing SR2ML

SR2ML is supported on three separate computing platforms: Linux, OSX (Apple Macintosh), and Microsoft Windows. Currently, SR2ML is downloadable from the SR2ML GitLab repository: `https://hpcgitlab.hpc.inl.gov/RAVEN_PLUGINS/SR2ML.git`. New users should contact SR2ML developers to get started. This typically involves the following steps:

- *Download SR2ML*
  You can download the source code from `https://github.com/idaholab/SR2ML.git`.

- *Use as a RAVEN Plugin*,
  RAVEN must first be downloaded from `https://github.com/idaholab/raven.git`.
  Detailed instructions are available from `https://github.com/idaholab/raven/wiki`. To register a plugin with RAVEN and make its components accessible, run the script:

  ```
  raven/scripts/install_plugins.py -s /abs/path/to/SR2ML
  ```

  After the plugin registration, follow the installation instructions at `https://github.com/idaholab/raven/wiki/installationMain` to install the required dependencies.

1.2 Accessing SR2ML from within RAVEN

The SR2ML can be accessed as a special subtype of the External model. The syntax is given in Listing 1.

Listing 1. Call SR2ML.FTModel from RAVEN input.

```
<ExternalModel name="FaultTree" subType="SR2ML.FTModel">
  <variables> Input and output variables needed by SR2ML
     </variables>
  ...
</ExternalModel>
```

1.3 User Manual Formats

In this manual, we employ the following formats to highlight certain parts with particular meanings (i.e., input structure, examples, and terminal commands):

- ***Python Coding:***

```
class AClass():
  def aMethodImplementation(self):
    pass
```

- ***SR2ML XML input example:***

```
<MainXMLBlock>
  ...
  <aXMLnode anAttribute='aValue'>
     <aSubNode>body</aSubNode>
  </aXMLnode>
  <!-- This is  commented block -->
  ...
</MainXMLBlock>
```

- ***Bash Commands:***

```
cd path/to/SR2ML/
cd ../../
```

1.4 Capabilities of SR2ML

This document provides a detailed description of the SR2ML plugin for the RAVEN code [1, 2]. The features included in this plugin are:

- Event Tree (ET) Model (see Section 2)
- Fault Tree (FT) Model (see Section 3)
- Markov Model (see Section 4)
- Reliability Block Diagram (RBD) Model (see Section 5)
- MCSSolver (see Section 6)
- Reliability Models (see Section 7)
- Maintenance Models (see Section 8)
- Data Labeling (see Section 9)
- ET Data Importer (see Section 10)
- FT Data Importer (see Section 11)

2 Event Tree Model

This model is designed to read the structure of the event tree (ET) from the file and to import such Boolean logic structure as a RAVEN model [1,2]. The ET must be specified in: the OpenPSA format (https://github.com/open-psa). As an example, the ET of Figure 1 is translated in the OpenPSA format as shown in Listing 13 below:

Listing 2. ET of Figure 1 in the OpenPSA format.

```
<define-event-tree name="eventTree">
    <define-functional-event name="ACC"/>
    <define-functional-event name="LPI"/>
    <define-functional-event name="LPR"/>
    <define-sequence name="1"/>
    <define-sequence name="2"/>
    <define-sequence name="3"/>
    <define-sequence name="4"/>
    <initial-state>
        <fork functional-event="ACC">
            <path state="0">
                <fork functional-event="LPI">
                    <path state="0">
                        <fork functional-event="LPR">
                            <path state="0">
                                <sequence name="1"/>
                            </path>
                            <path state="+1">
                                <sequence name="2"/>
                            </path>
                        </fork>
                    </path>
                    <path state="+1">
                        <sequence name="3"/>
                    </path>
                </fork>
            </path>
            <path state="+1">
                <sequence name="4"/>
            </path>
        </fork>
    </initial-state>
</define-event-tree>
```

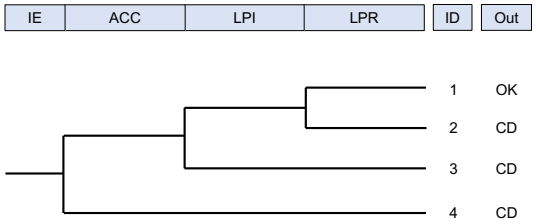


Figure 1. Example of event tree.

The ET of Figure 1 described in Listing 13 can be defined in the RAVEN input file as follows:

Listing 3. ET model input example.

```
<Models>
  ...
  <ExternalModel name="ET" subType="ETModel">
    <variables>
      statusACC,statusLPI,statusLPR,sequence
    </variables>
    <map var="statusACC">ACC</map>
    <map var="statusLPI">LPI</map>
    <map var="statusLPR">LPR</map>
    <sequenceID>sequence</sequenceID>
  </ExternalModel>
  ...
</Models>
```

All the specifications of the ET model are given in the **<ExternalModel>** block. Inside the **<ExternalModel>** block, the XML nodes that belong to this models are:

- **<variables>**, ***string, required parameter***, a list containing the names of both the input and output variables of the model
- **<sequenceID>**, ***string, required parameter***, the name of the alias variable that indicates the branch ID
- **<map>**, ***string, required parameter***, the name ID of the ET branching variable
  - **var**, ***required string attribute***, the ALIAS name ID of the ET branching variable.

Provided this definition and the ET model of Figure 1 and described in Listing 13, the resulting model in RAVEN is characterized by these variables:

- Input variables: statusACC, statusLPI, statusLPR
- Output variable: sequence

2.1 ET Model Reference Tests

- SR2ML/tests/test_ETmodel.xml
- SR2ML/tests/test_ETmodel_TD.xml

3 Fault Tree Model

This model is designed to read the structure of the fault tree (FT) from the file and to import such Boolean logic structures as a RAVEN model. The FT must be specified in the OpenPSA format (https://github.com/open-psa). As an example, the FT of Figure 2 is translated in the OpenPSA as shown below:

Listing 4. FT in OpenPSA format.

```
<opsa-mef>
    <define-fault-tree name="FT">
        <define-gate name="TOP">
            <or>
                <gate name="G1"/>
                <gate name="G2"/>
                <gate name="G3"/>
            </or>
        </define-gate>
        <define-component name="A">
            <define-gate name="G1">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE2"/>
                </and>
            </define-gate>
            <define-gate name="G2">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE3"/>
                </and>
            </define-gate>
            <define-basic-event name="BE1">
                <float value="1.2e-3"/>
            </define-basic-event>
            <define-component name="B">
                <define-basic-event name="BE2">
                    <float value="2.4e-3"/>
                </define-basic-event>
                <define-basic-event name="BE3">
                    <float value="5.2e-3"/>
                </define-basic-event>
            </define-component>
```

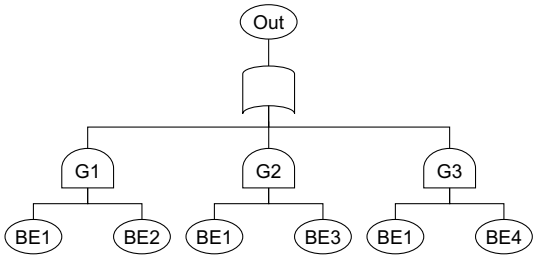


Figure 2. Example of FT.

```
        </define-component>
        <define-component name="C">
            <define-gate name="G3">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE4"/>
                </and>
            </define-gate>
            <define-basic-event name="BE4">
                <float value="1.6e-3"/>
            </define-basic-event>
        </define-component>
    </define-fault-tree>
</opsa-mef>
```

The FT of Figure 2 described in Listing 15 can be defined in the RAVEN input file as follows:

Listing 5. FT model input example.

```
  <Models>
    ...
    <ExternalModel name="FT" subType="FTModel">
      <variables>
        statusBE1,statusBE2,statusBE3,statusBE4,TOP
      </variables>
      <topEvents>TOP</topEvents>
      <map var="statusBE1">BE1</map>
      <map var="statusBE2">BE2</map>
      <map var="statusBE3">BE3</map>
      <map var="statusBE4">BE4</map>
    </ExternalModel>
```

```
    ...
  </Models>
```

All the specifications of the FT model are given in the **<ExternalModel>** block. Inside the **<ExternalModel>** block, the XML nodes that belong to this models are:

- **<variables>**, ***string, required parameter***, a list containing the names of both the input and output variables of the model
- **<topEvents>**, ***string, required parameter***, the name of the alias Top Event
- **<map>**, ***string, required parameter***, the name ID of the FT basic events
  - **var**, ***required string attribute***, the ALIAS name ID of the FT basic events

Provided this definition and the FT model of Figure 2 described in Listing 5, the resulting model in RAVEN is characterized by these variables:

- Input variables: statusBE1, statusBE2, statusBE3, statusBE4
- Output variable: TOP

3.1 FT Model Reference Tests

- SR2ML/tests/test_FTmodel.xml
- SR2ML/tests/test_FTmodel_TD.xml

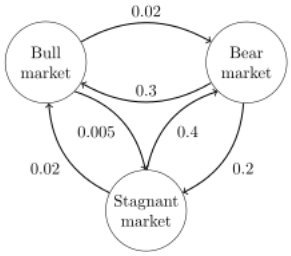


Figure 3. Example of a continuous time Markov chain (source Wikipedia: https://en.wikipedia.org/wiki/Markov_chain).

4 Markov Model

This model is designed to import a generic Markov chain as a RAVEN model. As an example, the Markov chain from Figure 3 is translated in the OpenPSA, as shown below:

Listing 6. Markov model input example.

```
<Models>
  <ExternalModel name="markov" subType="MarkovModel">
    <variables>initialState,finalState</variables>
    <initState>initialState</initState>
    <finState>finalState</finState>
    <endTime>1000</endTime>
    <state name="1"> <!-- Bull market -->
      <transition type="lambda" value="0.02" >2</transition>
      <transition type="lambda" value="0.005">3</transition>
    </state>
    <state name="2"> <!-- Bear market -->
      <transition type="lambda" value="0.3">1</transition>
      <transition type="lambda" value="0.2">3</transition>
    </state>
    <state name="3"> <!-- Stagnant market -->
      <transition type="lambda" value="0.02">1</transition>
      <transition type="lambda" value="0.4" >2</transition>
    </state>
  </ExternalModel>
</Models>
```

All the specifications of the Markov model are given in the **<ExternalModel>** block. Inside the **<ExternalModel>** block, the XML nodes that belong to this model are:

- **<variables>**, ***string, required parameter***, a list containing the names of both the input and output variables of the model
- **<initState>**, ***string, required parameter***, variable ID corresponding to the initial state
- **<finState>**, ***string, required parameter***, variable ID corresponding to the final state
- **<endTime>**, ***float, required parameter***, time horizon to evaluate the Markov chain transition history
- **<state>**, specifies a single node; inside a **<state>**, all possible transitions OUT of this state must be specified in the **<transition>** xml sub-nodes:
  - **transition**, ***required string attribute***, arrival state
  - **type**, ***required string attribute***, type of transition. Allowed transition types are: lambda, tau, instant, and unif (see below)
  - **value**, ***required string attribute***, value associated to the particular transition.

The following transition types are available:

- **lambda**: classical continuous time Markov chain transition rate in λ form
- **tau**: classical continuous time Markov chain transition rate in the $\tau = \frac{1}{\lambda}$ form
- **instant**: deterministic transition out of a particular state; the exact transition time is provided in an input
- **unif**: transition time is uniformly sampled between the two provided values in the **value** node.

4.1 Markov Model Reference Tests

- SR2ML/tests/test_markovModel_2states_tau.xml
- SR2ML/tests/test_markovModel_2states.xml
- SR2ML/tests/test_markovModel_3states_complexTrans.xml
- SR2ML/tests/test_markovModel_3states_instantTrans.xml
- SR2ML/tests/test_markovModel_3states.xml

5 Reliability Block Diagram Model

This model is designed to read the structure of the reliability block diagram (RBD) from the file and import such Boolean logic structures as a RAVEN model. The RBD must be specified in a specific format. As an example, the RBD of Figure 4 is translated in the RAVEN format as shown below:

Listing 7. RBD input file.

```
<Graph name="testGraph">
  <node name="CST">
    <childs>1</childs>
  </node>
  <node name="1">
    <childs>2,3,4</childs>
  </node>
  <node name="2">
    <childs>5</childs>
  </node>
  <node name="3">
    <childs>5</childs>
  </node>
  <node name="4">
    <childs>5</childs>
  </node>
  <node name="5">
    <childs>6,7,8</childs>
  </node>
  <node name="6">
    <childs>SG1</childs>
  </node>
  <node name="7">
    <childs>SG2</childs>
  </node>
  <node name="8">
    <childs>SG3</childs>
  </node>
  <node name="SG1">
  </node>
  <node name="SG2">
  </node>
  <node name="SG3">
```

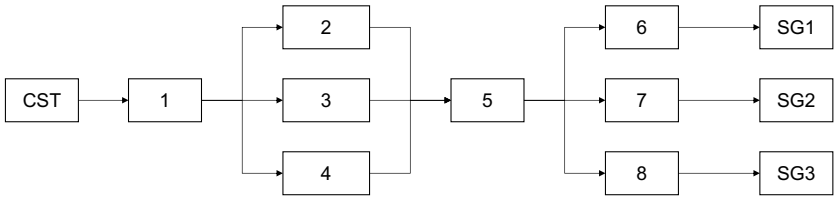


Figure 4. Example of RBD.

```
  </node>
</Graph>
```

The FT of RBD illustrated in Figure 4 and defined in Listing 7 can be defined in the RAVEN input file as follows:

Listing 8. RBD model input example.

```
  <Models>
    ...
    <ExternalModel name="graph" subType="GraphModel">
      <variables>
        status2,status3,status4,status5,
        statusSG1,statusSG2,statusSG3
      </variables>
      <modelFile>graphTest</modelFile>
      <nodesIN>CST</nodesIN>
      <nodesOUT>SG1,SG2,SG3</nodesOUT>
      <map var="status2">2</map>
      <map var="status3">3</map>
      <map var="status4">4</map>
      <map var="status5">5</map>
      <map var="statusSG1">SG1</map>
      <map var="statusSG2">SG2</map>
      <map var="statusSG3">SG3</map>
    </ExternalModel>
    ...
  </Models>
```

All the specifications of the RBD model are given in the **<ExternalModel>** block. Inside the **<ExternalModel>** block, the XML nodes that belong to this model are:

- **<variables>**, ***string, required parameter***, a list containing the names of both the input and output variables of the model
- **<modelFile>**, ***string, required parameter***, the name of the file that provide the RBD structure
- **<nodesIN>**, ***string, required parameter***, the name of the input nodes
- **<nodesOUT>**, ***string, required parameter***, the name of the output nodes
- **<map>**, ***string, required parameter***, the name ID of the RBD node
  - **var**, ***required string attribute***, the ALIAS name ID of the RBD node

Provided this definition, the RBD model of Figure 4 described in Listing 7, is the resulting model in RAVEN characterized by these variables:

- Input variables: status2, status3, status4, status5
- Output variable: statusSG1, statusSG2, statusSG3.

5.1 RBD Model Reference Tests

- SR2ML/tests/test_graphModel.xml
- SR2ML/tests/test_graphModel_TD.xml.

Table 1. MCS file format.

ID	Prob	MCS
1,	0.01,	BE1
2,	0.02,	BE3
3,	0.03,	BE2,BE4

6 MCSSolver

This model is designed to read from file a list of Minimal Cut Sets (MCSs) and to import such Boolean logic structure as a RAVEN model. Provided the sampled Basic Events (BEs) margin or probability values, the MCSSolver determines the margin or the probability of Top Event (TE). The list of MCS must be provided through a CSV file with the following format:

In this example:

- three MCSs are defined: MCS1 = BE1, MCS2 = BE3 and MCS3 = BE2 and BE4
- four BEs are defined: BE1, BE2, BE3 and BE4

Note that the MCSSolver considers only the list of MCSs and it discards the rest of data contained in the csv file.

All the specifications of the MCSSolver model are given in the **<ExternalModel>** block. Inside the **<ExternalModel>** block, the XML nodes that belong to this models are:

- **<variables>**, ***string, required parameter***, a list containing the names of both the input and output variables of the model
- **<solver>**
  - **type**, ***required string attribute***, type of calculation to be performed: probability or margin
  - **<solverOrder>**,***integer, required parameter***, solver order of $P(TE)$ for probability calculations: it specifies the maximum calculation envelope for $P(TE)$, i.e., the maximum number of MCSs to be considered when evaluating the probability of their union
  - **<metric>**,***string, required parameter***, metric used for margin calculations of the top event of $M(TE)$; it specifies the desired distance metric in terms of p values for the L_p norm (allowed values are: 0, 1, 2, inf).

- **<topEventID>**,***string, required parameter***, the name of the alias variable for the Top Event
- **<map>**,***string, required parameter***, the name ID of the ET branching variable
  - **var**, ***required string attribute***, the ALIAS name ID of the basic event

An example of RAVEN input file is the following:

Listing 9. MCSSolver model input example.

```
<Models>
  ...
  <Models>
    <ExternalModel name="MCSmodel" subType="SR2ML.MCSSolver">
      <variables>statusBE1,statusBE2,statusBE3,statusBE4,TOP</variables>
      <solver type='probability'>
        <solverOrder>3</solverOrder>
      </solver>
      <topEventID>TOP</topEventID>>
      <map var='pBE1'>BE1</map>
      <map var='pBE2'>BE2</map>
      <map var='pBE3'>BE3</map>
      <map var='pBE4'>BE4</map>
    </ExternalModel>
  </Models>
  ...
</Models>
```

In this case, the MCSs are written in terms of the variables BE1, BE2, BE3, and BE4. The values of the variables pBE1, pBE2, pBE3, and pBE4 (i.e., the probability values associated to each basic event) are generated outside the MCSSolver model (e.g., by a sampler) and passed to the MCSSolver model in order to calculate the probability value of the top event (e.g., the variable TOP in the example above). The **<map>** blocks allow the user to link the sampled probability value to the correspoding basic event.

If $solverOrder = 1$ then: $P(TE) = P(MCS1)+P(MCS2)+P(MCS3)$. If $solverOrder = 2$ then: $P(TE) = P(MCS1)+P(MCS2)+P(MCS3)-P(MCS1MCS2)-P(MCS1MCS3)-P(MCS2MCS3)$. If $solverOrder = 3$ then: $P(TE) = P(MCS1)+P(MCS2)+P(MCS3)-P(MCS1MCS2) - P(MCS1MCS3) - P(MCS2MCS3) + P(MCS1MCS2MCS3)$

6.1 Time dependent calculation

The MCSSolver can also perform time dependent calculation by providing in the MultiRun step a dataObject which contains the logic status of the Basic Events. This dataObject contains the logical status of the the Basic Event: 0 (Basic event set to False: probability=p(BE)) or 1 (Basic event set to True: probability=1.0). The format of the dataObject can be:

- HistorySet: it contains the temporal profile of each basic event (e.g., BE1, BE2, BE3, and BE4) as a time series (see test_MCSSolver_TD.xml). In this case it is needed to specify the ID of the time variable contained in the HistorySet in the **<timeID>** node.

Listing 10. Time dependent (from HistorySet) MCSSolver model input example.

```
<Models>
 <ExternalModel name="MCSmodel"
    subType="SR2ML.MCSSolver">
   <variables>statusBE1,statusBE2,statusBE3,statusBE4,TOP,time</va
   <solverOrder>3</solverOrder>
   <topEventID>TOP</topEventID>
   <timeID>time</timeID>
   <map var='statusBE1'>BE1</map>
   <map var='statusBE2'>BE2</map>
   <map var='statusBE3'>BE3</map>
   <map var='statusBE4'>BE4</map>
 </ExternalModel>
</Models>
```

- PointSet: it contains the interval time (initial and final time) under which each basic event is set to True (see test_MCSSolver_TD_fromPS.xml). As an example, the PointSet contains the IDs of the basic events in the and the initial and final time as shown below:

Listing 11. Example of PointSet for time dependent MCSSolver calculation.

```
<PointSet name="maintenanceSchedule_PointSet">
  <Input>BE</Input>
  <Output>tIn,tFin</Output>
</PointSet>
```

In this case, the MCSSolver requires the specification of the DataObject variables that contain the list of basic events (**<BE_ID>** node) and the initial and final time values (**<tInitial>** and **<tEnd>** nodes).

Listing 12. Time dependent (from PointSet) MCSSolver model input example.

```
<Models>
  <ExternalModel name="MCSmodel"
     subType="SR2ML.MCSSolver">
    <variables>pBE1,pBE2,pBE3,pBE4,TOP,time,BE1,BE2,BE3,BE4</variab
    <solverOrder>3</solverOrder>
    <BE_ID>BE</BE_ID>
    <tInitial>tIn</tInitial>
    <tEnd>tFin</tEnd>
    <topEventID>TOP</topEventID>
    <timeID>time</timeID>
    <map var='pBE1'>BE1</map>
    <map var='pBE2'>BE2</map>
    <map var='pBE3'>BE3</map>
    <map var='pBE4'>BE4</map>
  </ExternalModel>
</Models>
```

6.2 MCSSolver model reference tests

The following is the provided analytic test:

- test_MCSSolver.xml
- test_MCSSolver_TD.xml
- test_MCSSolver_TD_fromPS.xml

7 Reliability Models

Reliability Models are the most frequently used in life data analysis and reliability engineering. These models/functions give the probability of a component operating for a certain amount of time without failure. As such, the reliability models are function of time, in that every reliability value has an associated time value. In other words, one must specify a time value with the desired reliability value. This degree of flexibility makes the reliability model a much better reliability specification that the mean time to failure (MTTF), which only represents one point along the entire reliability model.

7.1 The Probability Density and Cumulative Density Functions

From probability and statistics, given a continuous random variable X, we denote:

- The probability density function (pdf), as $f(x)$
- The cumulative density function (cdf), as $F(x)$.

If x is a continuous random variable, then the probability of x takes on a value in the interval $[a, b]$ is the area under the pdf $f(x)$ from a to b

$$P(a \leq x \leq b) = \int_a^b f(x)dx \tag{1}$$

The cumulative distribution function is a function $F(x)$ of a random variable x and is defined for a number x_0 by

$$F(x_0) = P(x \leq x_0) = \int_{-\infty}^{x_0} f(s)ds \tag{2}$$

That is, for a given value x_0, $F(x_0)$ is the probability that the observed value of x would be, at most, x_0. The mathematical relationship between the pdf and cdf is given by:

$$F(x) = \int_{-\infty}^{x} f(s)ds \tag{3}$$

Conversely

$$f(x) = -\frac{dF(x)}{dx} \tag{4}$$

The functions most commonly used in reliability engineering and life data analysis, namely the reliability function and failure rate function, can be determined directly from the pdf definition, or $f(t)$. Different distributions exist, such as Lognormal, Exponential, Weibull, etc., and each of them has a predefined $f(t)$. These distributions were formulated by statisticians, mathematicians, and engineers to mathematically model or represent certain behavior. Some distributions tend to better represent life data and are most commonly referred to as lifetime distributions.

7.2 The Reliability and Failure Rate Models

Given the mathematical representation of a distribution, we can derive all functions needed for reliability analysis (i.e., reliability models/functions). This would only depend on the value of t after the value of the distribution parameters are estimated from data. Now, let T be the random variable defining the lifetime of the component with cdf $F(t)$, which is the time the component would operate before failure. The cdf $F(t)$ of the random variable T is given by

$$F(t) = \int_{-\infty}^{t} f(T)dT \tag{5}$$

If $F(t)$ is a differentiable function, the pdf $f(t)$ is given by

$$f(t) = -\frac{dF(t)}{dt} \tag{6}$$

The reliability function or survival function $R(t)$ of the component is given by

$$R(t) = P(T > t) = 1 - P(T \leq t) = 1 - F(t) \tag{7}$$

This is the probability that the component would operate after time t, sometimes called the survival probability. The failure rate of a system during the interval $[t, t+\Delta t]$ is the probability that a failure per unit time occurs in the interval, given that a failure has not occurred prior to t, the beginning of the interval. The failure rate function (i.e., instantaneous failure rate, conditional failure rate) of the hazard function is defined as the limit of the failure rate as the interval approaches zero

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{1}{R(t)} \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t} = \frac{1}{R(t)} \frac{dF(t)}{dt} = \frac{f(t)}{R(t)} \tag{8}$$

The failure rate function is the rate of change of the conditional probability of a failure at time t. It measures the likelihood that a component that has operated up until time t fails in the next instance of time. Generally, $\lambda(t)$ is the one tabulated because it is measured experimentally and because it tends to vary less rapidly with time than the other parameters. When $\lambda(t)$ is given, all other three parameters, $F(t)$, $f(t)$, $R(t)$, can be computed as follows

$$R(t) = \exp(-\int_{0}^{t} \lambda(s)ds) \tag{9}$$

$$f(t) = \lambda(t)R(t) = \lambda(t)\exp(-\int_{0}^{t} \lambda(s)ds) \tag{10}$$

$$F(t) = 1 - R(t) = 1 - \exp(-\int_{0}^{t} \lambda(s)ds) \tag{11}$$

The mean time between failure (MTBF) can be obtained by finding the expected value of the random variable T, time to failure. Hence

$$MTBF = E(T) = \int_{0}^{\infty} tf(t)dt = \int_{0}^{\infty} R(t)dt \tag{12}$$

7.3 The Lifetime Distributions or Aging Models

We would consider several of the most useful reliability models based on different probability distributions for describing the failure of continuous operating devices, including:

- Exponential, model `type` is `'exponential'`
- Erlangian, model `type` is `'erlangian'`
- Gamma, model `type` is `'gamma'`
- Lognormal, model `type` is `'lognorm'`
- Fatigue Life, model `type` is `'fatiguelife'`
- Weibull, model `type` is `'weibull'`
- Exponential Weibull, model `type` is `'exponweibull'`
- Bathtub, model `type` is `'bathtub'`
- Power Law, model `type` is `'powerlaw'`
- Log Linear, model `type` is `'loglinear'`.

The specifications of these models must be defined within a RAVEN `<ExternalModel>`. This XML node accepts the following attributes:

- `name`, ***required string attribute***, user-defined identifier of this model. **Note:** As with other objects, this identifier can be used to reference this specific entity from other input blocks in the XML.
- `subType`, ***required string attribute***, defines which of the subtypes should be used. For reliability models, the user must use `'SR2ML.ReliabilityModel'` as the subtype.

In the reliability `<ExternalModel>` input block, the following XML subnodes are required:

- `<variable>`, ***string, required parameter***. Comma-separated list of variable names. Each variable name needs to match a variable used or defined in the reliability model or variable coming from another RAVEN entity (i.e. Samplers, DataObjects and Models). **Note:** For all the reliability models, the following outputs variables would be available. If the user added these output variables in the node `<variables>`, these variables would be also available to for use anywhere in the RAVEN input to refer to the reliability model output variables.

- **'pdf_f'**, variable contains the calculated pdf value or values at given time instance(s), (i.e., a series of times).
- **'cdf_F'**, variable contains the calculated cdf value or values at given time instance(s), (i.e., a series of times).
- **'rdf_R'**, variable contains the calculated reliability function (rdf) value or values at given time instance(s), (i.e., a series of times).
- **'frf_h'**, variable contains the calculated failure rate function (frf) value or values at given time instance(s), (i.e., a series of times).

Note: When the external model variables are defined, at run time, RAVEN initializes them and tracks their values during the simulation.

- **<ReliabilityModel>**, ***required parameter***. The node is used to define the reliability model, and it contains the following required XML attribute:
  - **type**, ***required string attribute***, user-defined identifier of the reliability model. **Note:** the types for different reliability models can be found at the beginning of this section.

  In addition, this node accepts several different subnodes representing the model parameters depending on the **type** of the reliability model. The common subnodes for all reliability models are:
  - **<Tm>**, ***string or float or comma-separated float, required parameter***. Time instance(s) that the reliability models would use to compute the pdf, cdf, rdf, and frf values. If a string was provided, the reliability model would treat it as an input variable that came from entities of RAVEN. In this case, the variable must be listed in the sub-node **<variables>** under **<ExternalModel>**.
  - **<Td>**, ***string or float, optional parameter***. The time that the reliability models start to be active. If a string was provided, the reliability model would treat it as an input variable that came from RAVEN entities. In this case, the variable must be listed in the sub-node **<variables>** under **<ExternalModel>**.
    Default: 0.

In addition, if the user wants to use the **alias** system, the following XML block can be input:

- **<alias>** ***string, optional field*** specifies an alias for any variable of interest in the input or output space for the ExternalModel. These aliases can be used anywhere in the RAVEN input to refer to the ExternalModel variables. In the body of this node, the user specifies the name of the variable that the ExternalModel is going to use (during its execution). The actual alias, usable throughout the RAVEN input, is instead defined in the **variable** attribute of this tag.
  The user can specify aliases for both the input and the output space. As a sanity check,

RAVEN requires an additional required attribute **type**. This attribute can be either "input" or "output." **Note:** The user can specify as many aliases as needed.
Default: None

Example XML (Bathtub Reliability Model):

```
<ExternalModel name="bathtub"
  subType="SR2ML.ReliabilityModel">
      <variables>cdf_F, pdf_f, rdf_R, frf_h,
        tm</variables>
      <!-- xml portion for this plugin only -->
      <ReliabilityModel type="bathtub">
            <!-- scale parameter -->
            <beta>1.</beta>
            <theta>1.0</theta>
            <!-- mission time -->
            <Tm>tm</Tm>
            <!-- shape parameter -->
            <alpha>1.0</alpha>
            <rho>0.5</rho>
            <!-- weight parameter -->
            <c>0.5</c>
      </ReliabilityModel>
      <!--  alias can be used to represent any
        input/output variables -->
      <alias variable='bathtub_F'
        type='output'>cdf_F</alias>
      <alias variable='bathtub_f'
        type='output'>pdf_f</alias>
      <alias variable='bathtub_R'
        type='output'>rdf_R</alias>
      <alias variable='bathtub_h'
        type='output'>frf_h</alias>
</ExternalModel>
```

7.3.1 The Lognormal Model

The probability density function of the lognormal is given by

$$f(T_m) = \frac{1}{\alpha T_m \sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\ln \frac{T_m - T_d}{\beta}}{\alpha}\right)^2\right) \tag{13}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta > 0$, β is the scale parameter, α is the shape parameter, and T_d is the location parameter. This model accepts the following additional sub-nodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
<ExternalModel name="lognorm"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
        <ReliabilityModel type="lognorm">
                <!-- scale parameter -->
                <beta>1.</beta>
                <!-- mission time -->
                <Tm>tm</Tm>
                <!-- shape parameter -->
                <alpha>1.</alpha>
        </ReliabilityModel>
</ExternalModel>
```

7.3.2 The Exponential Model

The probability density function of the exponential is given by

$$f(T_m) = \lambda \exp\left(-\lambda \left(T_m - T_d\right)\right) \tag{14}$$

where $T_m \geq T_d$, $T_d, \lambda > 0$, λ is the mean failure rate or the inverse of scale parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<lambda>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.

Example XML:

```
<ExternalModel name="exponential"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
        <!-- xml portion for this plugin only -->
        <ReliabilityModel type="exponential">
                <!-- mean failure rate -->
                <lambda>1.</lambda>
                <!-- mission time -->
                <Tm>tm</Tm>
        </ReliabilityModel>
</ExternalModel>
```

7.3.3 The Weibull Model

The probability density function of the Weibull is given by

$$f(T_m) = \frac{\alpha}{\beta}\left(\frac{T_m - T_d}{\beta}\right)^{\alpha-1} \exp\left(-\left(\frac{T_m - T_d}{\beta}\right)^{\alpha}\right) \tag{15}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta > 0$, and β is the scale parameter, α is the shape parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
<ExternalModel name="weibull"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
```

```
            <!-- xml portion for this plugin only -->
            <ReliabilityModel type="weibull">
                    <!-- scale parameter -->
                    <beta>1.</beta>
                    <!-- mission time -->
                    <Tm>tm</Tm>
                    <!-- time delay -->
                    <Td>2.0</Td>
                    <!-- shape parameter -->
                    <alpha>1.0</alpha>
            </ReliabilityModel>
        </ExternalModel>
```

7.3.4 The Erlangian Model

The probability density function of the Erlangian is given by

$$f(T_m) = \frac{\lambda\left(\lambda T_m\right)^{k-1}\exp\left(-\lambda T_m\right)}{(k-1)!} \tag{16}$$

where $T_m \geq T_d$, $T_d, \lambda > 0$, and λ is the inverse of scale parameter, k is positive integer that control the shape, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<k>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.

  Default: 1 **Note:** k is a positive integer. If a float was provided, a warning would be raised.

Example XML:

```
        <ExternalModel name="erlangian"
          subType="SR2ML.ReliabilityModel">
                <variables>cdf_F, pdf_f, rdf_R, frf_h,
                   tm</variables>
                <!-- xml portion for this plugin only -->
```

```
        <ReliabilityModel type="erlangian">
                <!-- mean failure rate -->
                <lambda>0.1</lambda>
                <!-- mission time -->
                <Tm>tm</Tm>
                <!-- shape parameter -->
                <k>2</k>
        </ReliabilityModel>
</ExternalModel>
```

7.3.5 The Gamma Model

The probability density function of the Gamma is given by

$$f(T_m) = \frac{\beta\left(\beta\left(T_m - T_d\right)\right)^{\alpha-1}\exp\left(-\beta\left(T_m - T_d\right)\right)}{\Gamma\left(\alpha\right)} \tag{17}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta > 0$, and β is the inverse of scale parameter, α is the shape parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
<ExternalModel name="gamma"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
        <!-- xml portion for this plugin only -->
        <ReliabilityModel type="gamma">
                <!-- rate parameter -->
                <beta>0.1</beta>
                <!-- mission time -->
```

```
                <Tm>tm</Tm>
                <!-- shape parameter -->
                <alpha>2.</alpha>
            </ReliabilityModel>
        </ExternalModel>
```

7.3.6 The Fatigue Life Model (Birnbaum-Saunders)

The probability density function of the fatigue life is given by

$$f(T_m) = \frac{\frac{T_m - T_d}{\beta} + 1}{2\alpha\sqrt{2\pi\left(\frac{T_m - T_d}{\beta}\right)^3}} \exp\left(-\frac{\left(\frac{T_m - T_d}{\beta} - 1\right)^2}{2\left(\frac{T_m - T_d}{\beta}\right)\alpha^2}\right) \tag{18}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta > 0$, and β is the scale parameter, α is the shape parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
        <ExternalModel name="fatiguelife"
           subType="SR2ML.ReliabilityModel">
                <variables>cdf_F, pdf_f, rdf_R, frf_h,
                   tm</variables>
                <!-- xml portion for this plugin only -->
                <ReliabilityModel type="fatiguelife">
                        <!-- scale parameter -->
                        <beta>1.</beta>
                        <!-- mission time -->
                        <Tm>tm</Tm>
                        <!-- shape parameter -->
```

```
                <alpha>1.0</alpha>
            </ReliabilityModel>
        </ExternalModel>
```

7.3.7 The Exponentiated Weibull Model

The probability density function of the exponentiated Weibull is given by

$$f(T_m) = \gamma\alpha\left(1-\exp\left(-\left(\frac{T_m-T_d}{\beta}\right)^{\alpha}\right)\right)^{\gamma-1}\left(\frac{T_m-T_d}{\beta}\right)^{\alpha-1}\exp\left(-\left(\frac{T_m-T_d}{\beta}\right)^{\alpha}\right) \quad (19)$$

where $T_m \geq T_d$, $T_d, \alpha, \beta, \gamma > 0$, and β is the scale parameter, α and γ is the shape parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1
- **<gamma>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.

Example XML:

```
        <ExternalModel name="exponweibull"
           subType="SR2ML.ReliabilityModel">
                <variables>cdf_F, pdf_f, rdf_R, frf_h,
                   tm</variables>
                <!-- xml portion for this plugin only -->
                <ReliabilityModel type="exponweibull">
                        <!-- scale parameter -->
                        <beta>1.</beta>
                        <!-- mission time -->
                        <Tm>tm</Tm>
```

```
                    <!-- time delay -->
                    <Td>2.0</Td>
                    <!-- shape parameter -->
                    <alpha>1.0</alpha>
                    <gamma>0.5</gamma>
                </ReliabilityModel>
            </ExternalModel>
```

7.3.8 The Bathtub Model

The reliability function is given by:

$$R(T_m) = \exp\left(-c\beta\left(\frac{T_m - T_d}{\beta}\right)^{\alpha} - (1-c)\left(\exp\left(\frac{T_m - T_d}{\theta}\right)^{\rho} - 1\right)\right) \tag{20}$$

The failure rate function is given by

$$\lambda(T_m) = c\alpha\left(\frac{T_m - T_d}{\beta}\right)^{\alpha-1} + (1-c)\rho\left(\frac{T_m - T_d}{\theta}\right)^{\rho-1} \tag{21}$$

The probability density function of the Bathtub is given by

$$f(T_m) = \lambda(T_m)R(T_m) \tag{22}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta, \theta, \rho, c > 0$, and β, θ are the scale parameters, α, ρ are the shape parameters, $c \in [0, 1]$ is the weight parameter, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, required parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1
- **<theta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

- **<rho>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

- **<c>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.

  Default: 1

Example XML:

```
<ExternalModel name="bathtub"
  subType="SR2ML.ReliabilityModel">
      <variables>cdf_F, pdf_f, rdf_R, frf_h,
        tm</variables>
      <!-- xml portion for this plugin only -->
      <ReliabilityModel type="bathtub">
          <!-- scale parameter -->
          <beta>1.</beta>
          <theta>1.0</theta>
          <!-- mission time -->
          <Tm>tm</Tm>
          <!-- shape parameter -->
          <alpha>1.0</alpha>
          <rho>0.5</rho>
          <!-- weight parameter -->
          <c>0.5</c>
      </ReliabilityModel>
</ExternalModel>
```

7.3.9 The Power Law Model for Failure Rate Function

The hazard rate satisfies a power law as a function of time

$$\lambda(T_m) = \lambda + \alpha(T_m - T_d)^{\beta} \tag{23}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta, \lambda > 0$, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

- **<lambda>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
<ExternalModel name="powerlaw"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
        <!-- xml portion for this plugin only -->
        <ReliabilityModel type="powerlaw">
                <beta>1.0</beta>
                <alpha>1.0</alpha>
                <lambda>0.5</lambda>
                <Tm>tm</Tm>
        </ReliabilityModel>
</ExternalModel>
```

7.3.10 The Log Linear Model for Failure Rate Function

The hazard rate satisfies a exponential law as a function of time:

$$\lambda(T_m) = \exp\left(\alpha + \beta(T_m - T_d)\right) \tag{24}$$

where $T_m \geq T_d$, $T_d, \alpha, \beta > 0$, and T_d is the location parameter. This model accepts the following additional subnodes:

- **<alpha>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

- **<beta>**, ***string or float, optional parameter***. See the above definition. If a string was provided, the reliability model would treat it as an input variable that came from other RAVEN entity. In this case, the variable must be listed in the subnode **<variables>** under **<ExternalModel>**.
  Default: 1

Example XML:

```
<ExternalModel name="loglinear"
   subType="SR2ML.ReliabilityModel">
        <variables>cdf_F, pdf_f, rdf_R, frf_h,
           tm</variables>
        <!-- xml portion for this plugin only -->
        <ReliabilityModel type="loglinear">
                <beta>1.</beta>
                <alpha>1.</alpha>
                <Tm>tm</Tm>
        </ReliabilityModel>
</ExternalModel>
```

7.4 Reliability Models Reference Tests

- SR2ML/tests/reliabilityModel/test_bathtub.xml
- SR2ML/tests/reliabilityModel/test_erlangian.xml
- SR2ML/tests/reliabilityModel/test_expon.xml
- SR2ML/tests/reliabilityModel/test_exponweibull.xml
- SR2ML/tests/reliabilityModel/test_fatiguelife.xml
- SR2ML/tests/reliabilityModel/test_gamma.xml
- SR2ML/tests/reliabilityModel/test_loglinear.xml
- SR2ML/tests/reliabilityModeltest_lognorm.xml

- SR2ML/tests/reliabilityModel/test_normal.xml
- SR2ML/tests/reliabilityModel/test_powerlaw.xml
- SR2ML/tests/reliabilityModeltest_weibull.xml
- SR2ML/tests/reliabilityModel/test_time_dep_ensemble_reliability.xml.

8 Maintenance Models

Maintenance Models are models designed to model maintenance and testing from a reliability perspective. These models are designed to optimize preventive maintenance at the system level.

Two classes of models are considered here:

- Operating, i.e. model `type` is `'Operating'`
- Standby, i.e. model `type` is `'Standby'`

The specifications of these models must be defined within a RAVEN `<ExternalModel>`. This XML node accepts the following attributes:

- `name`, ***required string attribute***, user-defined identifier of this model. **Note:** As with other objects, this identifier can be used to reference this specific entity from other input blocks in the XML.
- `subType`, ***required string attribute***, defines which of the subtypes should be used. For maintenance models, the user must use `'SR2ML.MaintenanceModel'` as subtype.

In the maintenance `<ExternalModel>` input block, the following XML subnodes are required:

- `<variable>`, ***string, required parameter***. Comma-separated list of variable names. Each variable name needs to match a variable used or defined in the maintenance model or variable coming from other RAVEN entities (i.e., Samplers, DataObjects, and Models). **Note:** For all the maintenance models, the following outputs variables would be available. If the user added these output variables in the node `<variables>`, these variables would be also available to for use anywhere in the RAVEN input to refer to the maintenance model output variables.
  - `'avail'`, variable that contains the calculated availability value
  - `'unavail'`, variable that contains the calculated unavailability value

  Note: When the external model variables are defined, at run time, RAVEN initializes them and tracks their values during the simulation.
- `<MaintenanceModel>`, ***required parameter***. The node is used to define the maintenance model, and it contains the following required XML attribute:
  - `type`, ***required string attribute***, user-defined identifier of the maintenance model. **Note:** the types for different maintenance models can be found at the beginning of this section.

In addition, if the user wants to use the **alias** system, the following XML block can be input:

- **`<alias>`** ***string, optional field*** specifies alias for any variable of interest in the input or output space for the ExternalModel. These aliases can be used anywhere in the RAVEN input to refer to the ExternalModel variables. In the body of this node, the user specifies the name of the variable that the ExternalModel is going to use (during its execution). The actual alias, usable throughout the RAVEN input, is instead defined in the **`variable`** attribute of this tag.
  The user can specify aliases for both the input and the output space. As a sanity check, RAVEN requires an additional required attribute **`type`**. This attribute can be either "input" or "output." **Note:** The user can specify as many aliases as needed.
  Default: None

8.1 Operating Model

For an operating model, the unavailability u is calculated as

$$u = lambda * Tr/(1.0 + lambda * Tr) + Tpm/Tm \tag{25}$$

where:

- lambda: is the component failure rate
- Tr: mean time to repair
- Tpm: mean time to perform preventive maintenance
- Tm: preventive maintenance interval

Example XML:

```
<ExternalModel name="PMmodelOperating"
   subType="SR2ML.MaintenanceModel">
  <variables>lambda,Tm,avail,unavail</variables>
  <MaintenanceModel type="PMModel">
    <type>operating</type>
    <Tr>24</Tr>
    <Tpm>10</Tpm>
    <lambda>lambda</lambda>
    <Tm>Tm</Tm>
  </MaintenanceModel>
</ExternalModel>
```

8.2 Standby Model

For an operating model, the unavailability u is calculated as:

$$u = rho + 0.5 * lambda * Ti + Tt/Ti + (rho + lambda * Ti) * Tr/Ti + Tpm/Tm \quad (26)$$

where:

- rho: failure probability per demand
- Ti: surveillance test interval
- Tr: mean time to repair
- Tt: test duration
- Tpm: mean time to perform preventive maintenance
- Tm: preventive maintenance interval
- lamb: component failure rate

Example XML:

```
<ExternalModel name="PMmodelStandby"
   subType="SR2ML.MaintenanceModel">
  <variables>lambda,Tm,Ti,avail,unavail</variables>
  <MaintenanceModel type="PMModel">
    <type>standby</type>
    <Tr>24</Tr>
    <Tpm>10</Tpm>
    <Tt>5</Tt>
    <rho>0.01</rho>
    <lambda>lambda</lambda>
    <Ti>Ti</Ti>
    <Tm>Tm</Tm>
  </MaintenanceModel>
</ExternalModel>
```

9 Data Labeling

The **DataLabeling** post-processor is specifically used to label the data stored in the DataObjects. It accepts two DataObjects, one DataObject (i.e., reference DataObject) with type **PointSet** is used to label the other target DataObjects (type can be either **PointSet** or **HistorySet**).

In order to use the *DataLabeling* PP, the user needs to set the `subType` of a `<PostProcessor>` node:

`<PostProcessor name='ppName' subType='SR2ML.DataLabeling'/>`.

Several sub-nodes are available:

- `<label>`, ***string, required field***, the label variable name in the reference DataObject. This variable will be used to label the target DataObject.
- `<variable>`, ***required, xml node***. In this node, the following attribute should be specified:
  - `name`, ***required, string attribute***, the variable name, which should be exist in the reference DataObject.

  and the following sub-node should also be specified:

  - `<Function>`, ***string, required field***, this function creates the mapping from target DataObject to the reference DataObject.
    * `class`, ***string, required field***, the class of this function (e.g. Functions)
    * `type`, ***string, required field***, the type of this function (e.g. external)

In order to use this post-processor, the users need to specify two different DataObjects, i.e.

```
<DataObjects>
  <PointSet name="ET_PS">
    <Input>ACC, LPI</Input>
    <Output>sequence</Output>
  </PointSet>
  <PointSet name="sim_PS">
    <Input>ACC_status, LPI_status</Input>
    <Output>out</Output>
  </PointSet>
</DataObjects>
```

The first data object "ET_PS" contains the event tree with input variables "ACC, LPI" and output label "sequence". This data object will be used to classify the data in the second data object

"sim_PS". The results will be stored in the output data object with the same label "sequence". Since these two data objects contain different inputs, **<Functions>** will be used to create the maps between the inputs:

```
<Functions>
  <External file="func_ACC.py" name="func_ACC">
    <variable>ACC_status</variable>
  </External>
  <External file="func_LPI.py" name="func_LPI">
    <variable>LPI_status</variable>
  </External>
</Functions>
```

The inputs to these functions are the data from the target DataObject, and the outputs of these functions are the data from the reference DataObject.

Example Python Function for "func_ACC.py"

```
def evaluate(self):
  return self.ACC_status
```

Example Python Function for "func_LPI.py"

```
def evaluate(self):
  return self.LPI_status
```

Note: All the functions that are used to create the maps should be include the "evaluate" method.

The example of **DataLabeling** post processor is provided below:

```
<PostProcessor name="ET_Classifier"
   subType="SR2ML.DataLabeling">
  <label>sequence</label>
  <variable name='ACC'>
    <Function class="Functions"
       type="External">func_ACC</Function>
  </variable>
  <variable name='LPI'>
    <Function class="Functions"
       type="External">func_LPI</Function>
  </variable>
</PostProcessor>
```

The definitions for the XML nodes can be found in the RAVEN user manual. The label "sequence" and the variables "ACC, LPI" should be exist in the reference data object, while the functions "func_ACC, func_LPI" are used to map relationships between the reference and target data objects.

The labeling process can be achieved via the **<Steps>** as shown below:

```
<Simulation>
 ...
  <Steps>
    <PostProcess name="classify">
      <Input   class="DataObjects"  type="PointSet"
         >ET_PS</Input>
      <Input   class="DataObjects"  type="PointSet"
         >sim_PS</Input>
      <Model   class="Models"       type="PostProcessor"
         >ET_Classifier</Model>
      <Output  class="DataObjects"  type="PointSet"
         >sim_PS</Output>
    </PostProcess>
  </Steps>
 ...
</Simulation>
```

9.1 Data Labeling Reference Tests

- test_DataLabeling_postprocessor.xml
- test_DataLabeling_postprocessor_HS.xml.

10 Event Tree Data Importer

The **ETImporter** post-processor has been designed to import Event-Tree (ET) object into RAVEN. Since several ET file formats are available, as of now only the OpenPSA format (see https://open-psa.github.io/joomla1.5/index.php.html) is supported. As an example, the OpenPSA format ET is shown below:

Listing 13. ET in OpenPSA format.

```
<define-event-tree name="eventTree">
    <define-functional-event name="ACC"/>
    <define-functional-event name="LPI"/>
    <define-functional-event name="LPR"/>
    <define-sequence name="1"/>
    <define-sequence name="2"/>
    <define-sequence name="3"/>
    <define-sequence name="4"/>
    <initial-state>
        <fork functional-event="ACC">
            <path state="0">
                <fork functional-event="LPI">
                    <path state="0">
                        <fork functional-event="LPR">
                            <path state="0">
                                <sequence name="1"/>
                            </path>
                            <path state="+1">
                                <sequence name="2"/>
                            </path>
                        </fork>
                    </path>
                    <path state="+1">
                        <sequence name="3"/>
                    </path>
                </fork>
            </path>
            <path state="+1">
                <sequence name="4"/>
            </path>
        </fork>
    </initial-state>
</define-event-tree>
```

This is performed by saving the structure of the ET (from file) as a **PointSet** (only **PointSet** are allowed), since an ET is a static Boolean logic structure. Each realization in the **PointSet** represents a unique accident sequence of the ET, and the **PointSet** is structured as follows:

- Input variables of the **PointSet** are the branching conditions of the ET. The value of each input variable can be:
  - 0: event did occur (typically upper branch)
  - 1: event did not occur (typically lower branch)
  - -1: event is not queried (no branching occured)
- Output variables of the **PointSet** are the ID of each branch of the ET (i.e., positive integers greater than 0)

Note: that the 0 or 1 values are specified in the `<path state="0">` or `<path state="1">` nodes in the ET OpenPSA file.

Provided this definition, the ET described in Listing 13 will be converted to **PointSet** that is characterized by the following variables:

- Input variables: statusACC, statusLPI, statusLPR
- Output variable: sequence

and the corresponding **PointSet** if the `<expand>` node is set to False is shown in Table 2. If `<expand>` set to True, the corresponding **PointSet** is shown in Table 3.

Table 2. PointSet generated by RAVEN by employing the ET Importer Post-Processor with `<expand>` set to False for the ET of Listing 13.

ACC	LPI	LPR	sequence
0.	0.	0.	1.
0.	0.	1.	2.
0.	1.	-1.	3.
1.	-1.	-1.	4.

The ETImporter PP supports also:

- links to sub-trees **Note:** If the ET is split in two or more ETs (and thus one file for each ET), then it is only required to list all files in the Step. RAVEN automatically detect links among ETs and merge all of them into a single PointSet.

Table 3. PointSet generated by RAVEN by employing the ET Importer Post-Processor with **<expand>** set to True for the ET of Listing 13.

ACC	LPI	LPR	sequence
0.	0.	0.	1.
0.	0.	1.	2.
0.	1.	0.	3.
0.	1.	1.	3.
1.	0.	0.	4.
1.	0.	1.	4.
1.	1.	0.	4.
1.	1.	1.	4.

- by-pass branches
- symbolic definition of outcomes: typically outcomes are defined as either 0 (upper branch) or 1 (lower branch). If instead the ET uses the **success/failure** labels, then they are converted into 0/1 labels **Note:** If the branching condition is not binary or **success/failure**, then the ET Importer Post-Processor just follows the numerical value of the **<state>** attribute of the **<<path>>** node in the ET OpenPSA file.
- symbolic/numerical definition of sequences: if the ET contains a symbolic sequence then a .xml file is generated. This file contains the mapping between the sequences defined in the ET and the numerical IDs created by RAVEN. The file name is the concatenation of the ET name and "_mapping". As an example the file "eventTree_mapping.xml" generated by RAVEN:

```
<map Tree="eventTree">
  <sequence ID="0">seq_1</sequence>
  <sequence ID="1">seq_2</sequence>
  <sequence ID="2">seq_3</sequence>
  <sequence ID="3">seq_4</sequence>
</map>
```

  contains the mapping of four sequences defined in the ET (seq_1,seq_2,seq_3,seq_4) with the IDs generated by RAVEN (0,1,2,3). Note that if the sequences defined in the ET are both numerical and symbolic then they are all mapped.

- The ET can contain a branch that is defined as a separate block in the **<define-branch>** node and it is replicated in the ET; in such case RAVEN automatically replicate such branch when generating the PointSet.

The **<collect-formula>** are not considered since this node is used to connect the Boolean formulae generated by the Fault-Trees to the branch (i.e., fork) point.

In order to use the *ETImporter* PP, the user needs to set the **subType** of a **<PostProcessor>** node:

<PostProcessor name='ppName' subType='SR2ML.ETImporter'/>.

Several sub-nodes are available:

- **<fileFormat>**, ***string, required field***, specifies the format of the file that contains the ET structure (supported format: OpenPSA).
- **<expand>**,***bool, required parameter***, expand the ET branching conditions for all branches even if they are not queried

Example:

Listing 14. ET Importer input example.

```
<Files>
  <Input name="eventTreeTest" type="">eventTree.xml</Input>
</Files>

<Models>
  ...
  <PostProcessor name="ETimporter" subType="SR2ML.ETImporter">
    <fileFormat>OpenPSA</fileFormat>
    <expand>False</expand>
  </PostProcessor>
  ...
</Models>

<Steps>
  ...
  <PostProcess name="import">
    <Input   class="Files"        type=""
       >eventTreeTest</Input>
    <Model   class="Models"       type="PostProcessor"
       >ETimporter</Model>
    <Output  class="DataObjects"  type="PointSet"
       >ET_PS</Output>
  </PostProcess>
```

```
  ...
</Steps>

<DataObjects>
  ...
  <PointSet name="ET_PS">
    <Input>ACC,LPI,LPR</Input>
    <Output>sequence</Output>
  </PointSet>
  ...
</DataObjects>
```

10.1 ET Importer Reference Tests

- test_ETimporter.xml
- test_ETimporterMultipleET.xml
- test_ETimporterSymbolic.xml
- test_ETimporter_expand.xml
- test_ETimporter_DefineBranch.xml
- test_ETimporter_3branches.xml
- test_ETimporter_3branches_NewNumbering.xml
- test_ETimporter_3branches_NewNumbering_expanded.xml.

11 Fault Tree Data Importer

The **FTImporter** post-processor has been designed to import Fault-Tree (FT) object into RAVEN. Since several FT file formats are available, as of now only the OpenPSA format (see https://open-psa.github.io/joomla1.5/index.php.html) is supported. As an example, the FT in OpenPSA format is shown in Listing 15.

Listing 15. FT in OpenPSA format.

```
<opsa-mef>
    <define-fault-tree name="FT">
        <define-gate name="TOP">
            <or>
                <gate name="G1"/>
                <gate name="G2"/>
                <gate name="G3"/>
            </or>
        </define-gate>
        <define-component name="A">
            <define-gate name="G1">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE2"/>
                </and>
            </define-gate>
            <define-gate name="G2">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE3"/>
                </and>
            </define-gate>
            <define-basic-event name="BE1">
                <float value="1.2e-3"/>
            </define-basic-event>
            <define-component name="B">
                <define-basic-event name="BE2">
                    <float value="2.4e-3"/>
                </define-basic-event>
                <define-basic-event name="BE3">
                    <float value="5.2e-3"/>
                </define-basic-event>
            </define-component>
```

```
        </define-component>
        <define-component name="C">
            <define-gate name="G3">
                <and>
                    <basic-event name="BE1"/>
                    <basic-event name="BE4"/>
                </and>
            </define-gate>
            <define-basic-event name="BE4">
                <float value="1.6e-3"/>
            </define-basic-event>
        </define-component>
    </define-fault-tree>
</opsa-mef>
```

This is performed by saving the structure of the FT (from file) as a **PointSet** (only **PointSet** are allowed).

Each Point in the PointSet represents a unique combination of the basic events. The PointSet is structured as follows: input variables are the basic events, output variable is the top event of the FT. The value for each input and output variable can have the following values:

- 0: False
- 1: True

Provided this definition, the FT model of Listing 15 can be converted to **PointSet** that is characterized by these variables:

- Input variables: BE1, BE2, BE3, BE4
- Output variable: out

and it is structured is shown in Table 4.

In order to use the *FTImporter* PP, the user needs to set the `subType` of a `<PostProcessor>` node:

`<PostProcessor name='ppName' subType='SR2ML.FTImporter'/>`.

Several sub-nodes are available:

Table 4. PointSet generated by RAVEN by employing the FT Importer Post-Processor for the FT of Listing 15.

BE1	BE2	BE3	BE4	TOP
0.	0.	0.	0.	0.
0.	0.	0.	1.	0.
0.	0.	1.	0.	0.
0.	0.	1.	1.	0.
0.	1.	0.	0.	0.
0.	1.	0.	1.	0.
0.	1.	1.	0.	0.
0.	1.	1.	1.	0.
1.	0.	0.	0.	0.
1.	0.	0.	1.	1.
1.	0.	1.	0.	1.
1.	0.	1.	1.	1.
1.	1.	0.	0.	1.
1.	1.	0.	1.	1.
1.	1.	1.	0.	1.
1.	1.	1.	1.	1.

- **<fileFormat>**, ***string, required field***, specifies the format of the file that contains the FT structure (supported format: OpenPSA).
- **<topEventID>**,***string, required parameter***, the name of the top event of the FT

The example of FTImporter PostProcessor is shown in Listing 16

Listing 16. FT Importer input example.

```
<Files>
  <Input name="faultTreeTest"
     type="">FTimporter_not.xml</Input>
</Files>

<Models>
  ...
  <PostProcessor name="FTimporter" subType="SR2ML.FTImporter">
    <fileFormat>OpenPSA</fileFormat>
    <topEventID>TOP</topEventID>
  </PostProcessor>
  ...
```

```
</Models>

<Steps>
  ...
  <PostProcess name="import">
    <Input   class="Files"         type=""
       >faultTreeTest</Input>
    <Model   class="Models"        type="PostProcessor"
       >FTimporter</Model>
    <Output  class="DataObjects"   type="PointSet"
       >FT_PS</Output>
  </PostProcess>
  ...
</Steps>

<DataObjects>
  ...
  <PointSet name="FT_PS">
    <Input>BE1,BE2,BE3,BE4</Input>
    <Output>TOP</Output>
  </PointSet>
  ...
</DataObjects>
```

Important notes and capabilities:

- If the FT is split in two or more FTs (and thus one file for each FT), then it is only required to list all files in the Step. RAVEN automatically detect links among FTs and merge all of them into a single PointSet.
- Allowed gates: AND, OR, NOT, ATLEAST, CARDINALITY, IFF, imply, NAND, NOR, XOR
- If an house-event is defined in the FT:

Listing 17. FT Importer input example: house-event.

```
<opsa-mef>
    <define-fault-tree name="FT">
        <define-gate name="TOP">
            <or>
                <basic-event name="BE1"/>
```

```
                <basic-event name="BE2"/>
                <house-event name="HE1"/>
            </or>
        </define-gate>
        <define-house-event name="HE1">
                <constant value="true"/>
        </define-house-event>
    </define-fault-tree>
</opsa-mef>
```

then the HE1 is not part of the PointSet (value is fixed)

11.1 FT Importer Reference Tests

- test_FTimporter_and_withNOT_embedded.xml
- test_FTimporter_and_withNOT_withNOT_embedded.xml
- test_FTimporter_and_withNOT.xml
- test_FTimporter_and.xml
- test_FTimporter_atleast.xml
- test_FTimporter_cardinality.xml
- test_FTimporter_component.xml
- test_FTimporter_doubleNot.xml
- test_FTimporter_iff.xml
- test_FTimporter_imply.xml
- test_FTimporter_multipleFTs.xml
- test_FTimporter_nand.xml
- test_FTimporter_nor.xml
- test_FTimporter_not.xml
- test_FTimporter_or_houseEvent.xml
- test_FTimporter_or.xml
- test_FTimporter_xor.xml.

12 MCSImporter

The **MCSImporter** post-processor has been designed to import Minimal Cut Sets (MCSs) into RAVEN. This post-processor reads a csv file which contain the list of MCSs and it save this list as a DataObject (i.e., a PointSet). The csv file is composed by three columns; the first contains the ID number of the MCS, the second one contains the MCS probability value, the third one lists all the Basic Events contained in the MCS. An example of csv file is shown in Table 5.

Table 5. Example of csv file which contains four MCSs.

ID,	Prob,	MCS,
1.,	1.8E-2,	D
2.,	4.0E-3,	B
3.,	3.0E-4,	A,C
4.,	2.1E-5,	E,C

The PointSet is structured to include all Basic Event, the MCS ID, the MCS probability, and the outcome of such MCS (always set to 1). MCS ID and MCS probability are copied directly from the csv file. For each MCS, the Basic Events can have two possible values:

- 0: Basic Event is not included in the MCS
- 1: Basic Event is included in the MCS

The PointSet generated from the csv file of Table 5 is shown in Table 6.

Table 6. PointSet generated by RAVEN for the list of MCSs shown in Table 5.

A	B	C	D	E	MCS_ID	probability	out
0	0	0	1	0	1	1.8E-2	1
0	1	0	0	0	2	4.0E-3	1
1	0	1	0	0	3	3.0E-4	1
0	0	1	0	1	4	4.0E-3	1

In order to use the *MCSImporter* PP, the user needs to set the `subType` of a `<PostProcessor>` node:

```
<PostProcessor name='ppName' subType='SR2ML.MCSImporter'/>.
```

Several sub-nodes are available:

- **<expand>**,***bool, required parameter***, expand the set of Basic Events by including all PRA Basic Events and not only the once listed in the MCSs
- **<BElistColumn>**,***string, optional parameter***, if expand is set to True, then this node contains the column of the csv file which contains all the PRA Basic Events

Example:

Listing 18. MCS Importer input example (no expand).

```
<Files>
  <Input name="MCSlistFile" type="MCSlist">MCSlist.csv</Input>
</Files>

<Models>
  <PostProcessor name="MCSImporter"
     subType="SR2ML.MCSImporter">
    <expand>False</expand>
  </PostProcessor>
</Models>

<Steps>
  <PostProcess name="import">
    <Input   class="Files"        type="MCSlist"
       >MCSlistFile</Input>
    <Model   class="Models"       type="PostProcessor"
       >MCSImporter</Model>
    <Output  class="DataObjects"  type="PointSet"
       >MCS_PS</Output>
  </PostProcess>
</Steps>

<DataObjects>
  <PointSet name="MCS_PS">
    <Input>A,B,C,D,E</Input>
    <Output>MCS_ID,probability,out</Output>
  </PointSet>
</DataObjects>
```

Example:

Listing 19. MCS Importer input example (expanded).

```
<Files>
  <Input name="MCSlistFile" type="MCSlist">MCSlist.csv</Input>
  <Input name="BElistFile"  type="BElist" >BElist.csv</Input>
</Files>

<Models>
  <PostProcessor name="MCSImporter"
     subType="SR2ML.MCSImporter">
    <expand>False</expand>
  </PostProcessor>
</Models>

<Steps>
  <PostProcess name="import">
    <Input   class="Files"         type="MCSlist"
       >MCSlistFile</Input>
    <Input   class="Files"         type="BElist"
       >BElistFile</Input>
    <Model   class="Models"        type="PostProcessor"
       >MCSimporter</Model>
    <Output  class="DataObjects"   type="PointSet"
       >MCS_PS</Output>
  </PostProcess>
</Steps>

<DataObjects>
  <PointSet name="MCS_PS">
    <Input>A,B,C,D,E,F,G</Input>
    <Output>MCS_ID,probability,out</Output>
  </PointSet>
</DataObjects>
```

13 Discrete Risk Measures

This Post-Processor calculates a series of risk importance measures from a PointSet. This calculation is performed for a set of input parameters given an output target.

The user is required to provide the following information:

- the set of input variables. For each variable the following need to be specified:
  - the set of values that imply a reliability value equal to 1 for the input variable
  - the set of values that imply a reliability value equal to 0 for the input variable
- the output target variable. For this variable it is needed to specify the values of the output target variable that defines the desired outcome.

The following variables are first determined for each input variable i:

- R_0 Probability of the outcome of the output target variable (nominal value)
- R_i^+ Probability of the outcome of the output target variable if reliability of the input variable is equal to 0
- R_i^- Probability of the outcome of the output target variable if reliability of the input variable is equal to 1

Available measures are:

- Risk Achievement Worth (RAW): $RAW = R_i^+/R_0$
- Risk Achievement Worth (RRW): $RRW = R_0/R_i^-$
- Fussell-Vesely (FV): $FV = (R_0 - R_i^-)/R_0$
- Birnbaum (B): $B = R_i^+ - R_i^-$

In order to use the *RiskMeasureDiscrete* PP, the user needs to set the **subType** of a **<PostProcessor>** node:

<PostProcessor name='ppName' subType='SR2ML.RiskMeasureDiscrete'/>.

Several sub-nodes are available:

In the **<PostProcessor>** input block, the following XML sub-nodes are required, independent of the **subType** specified:

- **<measures>**, ***string, required field***, desired risk importance measures that have to be computed (RRW, RAW, FV, B)
- **<variable>**, ***string, required field***, ID of the input variable. This node is provided for each input variable. This nodes needs to contain also these attributes:
  - **R0values**, ***float, required field***, interval of values (comma separated values) that implies a reliability value equal to 0 for the input variable
  - **R1values**, ***float, required field***, interval of values (comma separated values) that implies a reliability value equal to 1 for the input variable
- **<target>**, ***string, required field***, ID of the output variable. This nodes needs to contain also the attribute **values**, ***string, required field***, interval of values of the output target variable that defines the desired outcome

Example: This example shows an example where it is desired to calculate all available risk importance measures for two input variables (i.e., pumpTime and valveTime) given an output target variable (i.e., Tmax). A value of the input variable pumpTime in the interval $[0, 240]$ implies a reliability value of the input variable pumpTime equal to 0. A value of the input variable valveTime in the interval $[0, 60]$ implies a reliability value of the input variable valveTime equal to 0. A value of the input variables valveTime and pumpTime in the interval $[1441, 2880]$ implies a reliability value of the input variables equal to 1. The desired outcome of the output variable Tmax occurs in the interval $[2200, 2500]$.

```
<Simulation>
  ...
  <Models>
    ...
    <PostProcessor name="riskMeasuresDiscrete"
       subType="RiskMeasuresDiscrete">
      <measures>B,FV,RAW,RRW</measures>
      <variable R0values='0,240'
         R1values='1441,2880'>pumpTime</variable>
      <variable R0values='0,60'
         R1values='1441,2880'>valveTime</variable>
      <target   values='2200,2500'
         >Tmax</target>
    </PostProcessor>
    ...
  </Models>
  ...
</Simulation>
```

This Post-Processor allows the user to consider also multiple datasets (a data set for each initiating event) and calculate the global risk importance measures. This can be performed by:

- Including all datasets in the step

```
<Simulation>
  ...
  </Steps>
    ...
    <PostProcess name="PP">
      <Input   class="DataObjects"  type="PointSet"
         >outRun1</Input>
      <Input   class="DataObjects"  type="PointSet"
         >outRun2</Input>
      <Model   class="Models"       type="PostProcessor"
         >riskMeasuresDiscrete</Model>
      <Output  class="DataObjects"  type="PointSet"
         >outPPS</Output>
      <Output  class="OutStreams"   type="Print"
         >PrintPPS_dump</Output>
    </PostProcess>
  </Steps>
  ...
</Simulation>
```

- Adding in the Post-processor the frequency of the initiating event associated to each dataset

```
<Simulation>
  ...
  <Models>
    ...
    <PostProcessor name="riskMeasuresDiscrete"
       subType="SR2ML.RiskMeasuresDiscrete">
      <measures>FV,RAW</measures>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Astatus</variable>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Bstatus</variable>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Cstatus</variable>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Dstatus</variable>
      <target   values='0.9,1.1'>outcome</target>
```

```
      <data      freq='0.01'>outRun1</data>
      <data      freq='0.02'>outRun2</data>
    </PostProcessor>
    ...
  </Models>
  ...
</Simulation>
```

This post-processor can be made time dependent if a single HistorySet is provided among the other data objects. The HistorySet contains the temporal profiles of a subset of the input variables. This temporal profile can be only boolean, i.e., 0 (component offline) or 1 (component online). Note that the provided history set must contains a single History; multiple Histories are not allowed. When this post-processor is in a dynamic configuration (i.e., time-dependent), the user is required to specify an xml node **<temporalID>** that indicates the ID of the temporal variable. For each time instant, this post-processor determines the temporal profiles of the desired risk importance measures. Thus, in this case, an HistorySet must be chosen as an output data object. An example is shown below:

```
<Simulation>
  ...
  <Models>
    ...
    <PostProcessor name="riskMeasuresDiscrete"
       subType="SR2ML.RiskMeasuresDiscrete">
      <measures>B,FV,RAW,RRW,R0</measures>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Astatus</variable>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Bstatus</variable>
      <variable R1values='-0.1,0.1'
         R0values='0.9,1.1'>Cstatus</variable>
      <target    values='0.9,1.1'>outcome</target>
      <data      freq='1.0'>outRun1</data>
      <temporalID>time</temporalID>
    </PostProcessor>
    ...
  </Models>
  ...
  <Steps>
    ...
    <PostProcess name="PP">
      <Input      class="DataObjects"  type="PointSet"
         >outRun1</Input>
```

```
      <Input      class="DataObjects"  type="HistorySet"
         >timeDepProfiles</Input>
      <Model      class="Models"       type="PostProcessor"
         >riskMeasuresDiscrete</Model>
      <Output     class="DataObjects"  type="HistorySet"
         >outHS</Output>
      <Output     class="OutStreams"   type="Print"
         >PrintHS</Output>
    </PostProcess>
    ...
  </Steps>
  ...
</Simulation>
```

14 Margin Models

Margin Models are models designed to calculate the margin of a component for margin-based reliability calculations.

The classes of models considered here are as follows:

- PointSetMarginModel, i.e. model `type` is `'PointSetMarginModel'`

The specifications of these models must be defined within a RAVEN `<ExternalModel>`. This XML node accepts the following attributes:

- `name`, ***required string attribute***, user-defined identifier of this model. **Note:** As with other objects, this identifier can be used to reference this specific entity from other input blocks in the XML.
- `subType`, ***required string attribute***, defines which of the subtypes should be used. For margin models, the user must use `'SR2ML.MarginModel'` as subtype.

In the margin `<ExternalModel>` input block, the following XML subnodes are required:

- `<variable>`, ***string, required parameter***. Comma-separated list of variable names. Each variable name needs to match a variable used or defined in the matign model or variable coming from other RAVEN entities (i.e., Samplers, DataObjects, and Models).
- `<MarginModel>`, ***required parameter***. The node is used to define the maintenance model, and it contains the following required XML attribute:
  - `type`, ***required string attribute***, user-defined identifier of the margin model.

In addition, if the user wants to use the **alias** system, the following XML block can be input:

- `<alias>` ***string, optional field*** specifies alias for any variable of interest in the input or output space for the ExternalModel. These aliases can be used anywhere in the RAVEN input to refer to the ExternalModel variables. In the body of this node, the user specifies the name of the variable that the ExternalModel is going to use (during its execution). The actual alias, usable throughout the RAVEN input, is instead defined in the `variable` attribute of this tag.
  The user can specify aliases for both the input and the output space. As a sanity check, RAVEN requires an additional required attribute `type`. This attribute can be either "input" or "output." **Note:** The user can specify as many aliases as needed.
  Default: None

14.1 PointSetMarginModel Model

For the PointSetMarginModel model, in the margin **<MarginModel>** input block, the following XML subnodes are required:

- **<failedDataFileID>**, ***string, required parameter***. ID of the file containing the failure data to be used as reference data
- **<marginID>**, ***required parameter***. ID of the variable where the margin needs to be stored
- **<map>**, ***required parameter***. This node is used to map actual measured data dimension and the failure data dimension. In here, the ID of the Raven variable representing a dimension of the actual data
  - **var**, ***required string attribute***, the ID of the corresponding dimension in the failure data.

Example XML:

```
<ExternalModel name="PointSetMargin"
   subType="SR2ML.MarginModel">
  <variables>actualTime,actualTemp,marginPS1</variables>
  <MarginModel type="PointSetMarginModel">
    <failedDataFileID>failureData.csv</failedDataFileID>
    <marginID>marginPS1</marginID>
    <map var='time'>actualTime</map>
    <map var='avgT'>actualTemp</map>
  </MarginModel>
</ExternalModel>
```

Document Version Information

This document has been compiled using the following version of the plug-in git repository:
fc3c0d531b108b2f2017a3e0e12ed8b25d8641ba mandd Fri, 9 Jul 2021 16:39:49 -0600

References

[1] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, and J. Chen, "Raven user manual," Tech. Rep. INL/EXT-15-34123, March 2017.

[2] A. Alfonsi, C. Rabiti, D. Mandelli, J. Cogliati, C. Wang, P. W. Talbot, D. P. Maljovec, and C. Smith, "Raven theory manual," tech. rep., Idaho National Laboratory (INL), 2017.

APPENDIX B – LOGOS USER MANUAL

MANUAL

INL/EXT-20-61006
Revision 0
Printed July 12, 2021

LOGOS User Manual

Congjian Wang and Diego Mandelli

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

INL/EXT-20-61006
Revision 0
Printed July 12, 2021

LOGOS User Manual

Congjian Wang
Diego Mandelli

Contents

1 Introduction

Industry equipment reliability (ER) and asset management (AM) programs are essential elements that help ensure the safe and economical operation of nuclear power plants (NPPs). The effectiveness of these programs is addressed in several industry-developed and regulatory programs. However, these programs have proven labor-intensive and expensive. There is an opportunity to significantly enhance the collection, analysis, and use of this information in order to provide more cost-effective plant operation. LOGOS provides computational capabilities to optimize plant resources such as maintenance optimization (ER application) and optimal component replacement schedule (AM application) by using state-of-the-art discrete optimization methods.

LOGOS is a software package and RAVEN [1, 2] plugin that contains a set of discrete optimization models designed to solve capital budgeting optimization problems by integrating economic and reliability data into the analysis framework. More specifically, given system, structure and component (SSC) health (e.g., failure rate or probability), O&M costs, replacement costs, and cost associated with component failure and budget constraints, LOGOS provides the optimal set of projects (e.g., SSC replacement) to maximize profit and satisfy the provided reliability requirements. The aforementioned input data can be either deterministic or stochastic in nature (i.e., they can be point values or probability distribution functions). In the latter case, several scenarios are generated by sampling the provided distributions.

The developed models are based on different versions of the knapsack optimization problem. Two main classes of optimization models were initially developed: deterministic and stochastic. Stochastic optimization models evolve deterministic models by explicitly considering data uncertainties associated with constraints or item cost and reward. In FY-20, we moved forward by implementing two schedule optimization methods. The first one reformulates the capital budgeting problem in a distributionally robust form which allows the user to rely on data directly rather than proposing a distribution from the data itself. The second one reformulates the capital budgeting explicitly using risk measures as variable to be maximized or minimized.

These models can be employed as stand-alone models or interfaced with the INL-developed RAVEN code to propagate data uncertainties and analyze the generated data (i.e., sensitivity analysis).

1.1 Acquiring and Installing LOGOS

LOGOS is supported on three separate computing platforms: Linux, OSX (Apple Macintosh), and Microsoft Windows. Currently, LOGOS can be downloaded from the LOGOS GitLab repository: `https://hpcgitlab.hpc.inl.gov/RAVEN_PLUGINS/LOGOS.git`. New users should contact LOGOS developers to get started with LOGOS. This typically involves the following steps:

- *Download LOGOS*
  You can download the source code for LOGOS from `https://hpcgitlab.hpc.inl.gov/RAVEN_PLUGINS/LOGOS.git`.

- *Install LOGOS dependencies*

```
path/to/LOGOS/build.sh --install
```

- *Activate LOGOS Libraries*

```
source activate LOGOS_libraries
```

- *Test LOGOS*

```
python run_tests.py
```

  Alternatively, the `logos` script contained in the folder "`LOGOS`" can be directly used:

```
path/to/LOGOS/logos -i <inputFile.xml> -o <outputFile.csv>
```

- *For use as a RAVEN Plugin*, RAVEN must first be downloaded from `https://github.com/idaholab/raven.git`.
  Detailed instructions are available from `https://github.com/idaholab/raven/wiki`. To register a plugin with RAVEN and make its components accessible, run the script:

```
raven/scripts/install_plugins.py -s /abs/path/to/LOGOS
```

  After the plugin registration, then following the installation instruction at `https://github.com/idaholab/raven/wiki/installationMain` to install the required dependencies.

1.2 User Manual Formats

In this manual, we employ the following formats to highlight specific elements with particular meanings (i.e., input structure, examples, and terminal commands):

- ***Python Coding:***

```
class AClass():
  def aMethodImplementation(self):
    pass
```

- ***LOGOS XML input example:***

```
<MainXMLBlock>
  ...
  <aXMLnode anAttribute='aValue'>
     <aSubNode>body</aSubNode>
  </aXMLnode>
  <!-- This is  commented block -->
  ...
</MainXMLBlock>
```

- ***Bash Commands:***

```
cd path/to/LOGOS/
./build.sh --install
cd ../../
```

1.3 Components of LOGOS

In LOGOS, eXtensible Markup Language (XML) format is used to create the input file. For more information about XML, please click on the link: **XML tutorial**.
The main input blocks are as follows:

- **<Logos>**: The root node containing the entire input; all of the subsequent blocks fit inside the *LOGOS* block.
- **<Settings>**: Specifies the calculation settings (i.e. options for optimization solvers, options for constraints, and working directory.)
- **<Sets>**: Specifies a collection of data, possibly including numeric data (e.g. real or integer values) as well as symbolic data (e.g. strings) typically used to specify valid indices for indexed components. **Note:** numeric data provided in the **<Sets>** would be treated as strings.
- **<Parameters>**: Specifies a collection of parameters, which are numerical values used to formulate constraints and objectives in a optimization model. A parameter can denote a single value, an array of values, or a multi-dimensional array of values.
- **<Uncertainties>**: Specifies a collection of scenarios, which are numerical values used to simulate variations within parameters. A scenarios should follow the same format as the parameter.
- **<ExternalConstraints>**: Specifies a collection of external constraints, which are Python functions used to add additional constraints to the current optimization problem.

Each of these components is explained in dedicated sections of the user manual.

1.4 Capabilities of LOGOS

This document provides a detailed description of LOGOS. The features included in LOGOS are:

- Overview of modeling components (see Section 2)
- Deterministic capital budgeting (see Section 3)
- Prioritizing project selection to hedge against uncertainty (see Section 4)
- Distributionally robust optimization (see Section 5)
- Risk-based stochastic capital budgeting using conditional Value-at-Risk (See Section 6)
- Plugin for the RAVEN code (see Section 7)
- SSC cashflow and NPV models (see Section 8)

2 Overview of Modeling Components

We consider a capital budgeting problem for a nuclear generation station, with possible extension to a larger fleet of plants. Due to limited resources, we can only select a subset from a number of candidate investment projects. Our goal is to maximize overall net present value (NPV), or a variant of this objective, when we incorporate uncertainty into project cost and revenue streams. In doing so, we must respect resource limits and capture key structural and stochastic dependencies of the system. Example projects include upgrading a steam turbine, refurbishing or replacing a set of reactor coolant pumps, and replacing a set of feed-water heaters. Selecting an individual project has multiple facets and implications.

- **Rewards or Net Present Values**: Selecting a project can improve revenue (e.g., upgrading a steam turbine may lead to an uprate in plant capacity resulting in larger revenue from selling power.) Replacing a key system component can improve reliability, increasing revenue due to a reduction in forced outages as well as operations and maintenance (O&M) costs. Choosing to perform minimum maintenance versus refurbishing a component or replacing and improving a system can produce reward streams that can be negative or positive depending on the selection. Parameter **<net_present_values>** is used to specify the rewards (see Section 2.2).

- **Resources and Liabilities**: Critical resources, including (i) capital costs, (ii) O&M costs, (iii) time and labor-hours during a planned outage, and (iv) personnel, installation and maintenance of equipment, workspaces, etc.. Within these categories, resources can further subcategorized, (witch each subcategory having its own budget), according to the plant's organizational structure to provide multiple "colors" of money within capital costs, O&M costs, personnel availability, etc.. The set **<resources>** and parameter **<available_capitals>** are used to specify the resources and liabilities (see Section 2.1 and Section 2.2).

- **Costs**: Selecting a project in year t induces multiple cost streams in year t as well as in subsequent years; we interpret "cost" broadly to include commitment of critical resources. The parameter **<costs>** is used to specify the costs (see Section 2.2).

- **Time Periods**: Multiple capital projects can compete for same time period, limiting project selection. The set **<time_periods>** is used to provide indices for **costs** and **available capitals** (see Section 2.1).

- **Options**: The goal of selecting a project is typically to improve or maintain a particular function the plant performs, and there may be multiple ways to carry out the task. A project may be performed over a three-year period–say, years 't, $t+1$, $t+2$'–or the start of the project could instead be two years hence, changing the equation to '$t+2$, $t+3$, $t+4$'. Alternatively, at increased cost and benefit, it may be possible to complete the project in two years: 't, $t+1$' or '$t+2$, $t+3$'. When selecting a project to uprate plant capacity, we may have the options of increasing it by 3% or 6%. In all these cases, we can perform the

project in, at most, one particular way, out of a collection of options. We represent this by cloning a "project" into multiple project-option pairs, and adding a constraint saying that we can select, at most, one from this set of options. The set **<options>** is used to provided indices for these multiple project-option pairs (see Section 2.1).

- **Capitals**: If we consider maintenance for multiple units in an NPP in parallel, it has to be decided whether to accept a particular replacement and, in the positive case in which unit to conduct the corresponding replacement. In this case, the set **<capitals>** is used to provided indices for these units (see Section 2.1).

- **Available Capitals**: This is available budgets for resources/units. The parameter **<available capitals>** is used to specify the available capitals for different resources/units at different t (see Section 2.2).

- **Non-Selection**: Not selecting a project also has implications, inducing a growth in O&M costs in future years, a decrease in plant production, an increase in forced outages, and even risking a premature end to plant life. Thus, not selecting a project can be seen as one more "option" for how a larger project is executed, expanding the list discussed earlier. Selection is of the "do nothing" option is reflected in both liability streams and reward streams. This can be activated by setting **<nonSelection>** to **'True'** (see Section 2.5).

- **Uncertainty**: One limitation of traditional optimization models for capital budgeting is that they do not account for uncertainty in reward and cost streams associated with individual projects, nor do they account for uncertainty in resource availability in future years. Projects can incur cost over-runs, especially when projects are large, performed infrequently, or when there is uncertainty regarding technical viability, external contractors, and/or suppliers of requisite parts and materials. Occasionally, projects are performed ahead of schedule and with savings in cost. Planned budgets for capital improvements can be cut, and key personnel may be lost. Or, there may be surprise budgetary windfalls for maintenance activities due to decreased costs for "unplanned" maintenance. The XML node **<Uncertainties>** is used to specify such uncertainties (see Section 2.3).

LOGOS consists of a collection of modeling entities/components that define different aspects of the model, including **<Sets>**, **<Parameters>**, **<Uncertainties>**, and **<External Constraints>**. In addition, the **<Setting>** block specifies how the overall computation should run.

2.1 Sets

This subsection contains information regarding the XML nodes used to define the **<Sets>** of the optimization model being performed through LOGOS. **<Sets>** specifies a collection of data, possibly including numeric data (e.g. real or integer values) as well as symbolic data (e.g. strings)

typically used to specify the valid indices for indexed components. **Note:** Numeric data provided in **<Sets>** would be treated as strings. **<Sets>** accepts the following additional sub-nodes:

- **<investments>**, ***comma/space-separated string, required***, specifies the valid indices for investment projects.
- **<capitals>**, ***comma/space-separated string, optional***, specifies the indices for NPP units.
- **<time_periods>**, ***comma/space-separated string, optional***, specifies the indices for time.
- **<resources>**, ***comma/space-separated string, optional***, specifies indices for the resources and liabilities.
- **<options>**, ***semi-colon separated list of strings, optional***, specifies the indices for multiple project-option pairs. This sub-node accepts the following attribute:
  - **index**, ***string, required***, specifies the index dependence. Valid index is **'investments'**.

Example XML:

```
<Sets>
  <investments>
      HPFeedwaterHeaterUpgrade
      PresurizerReplacement
      ...
      ReplaceMoistureSeparatorReheater
  </investments>
  <time_periods>year1 year2 year3 year4 year5</time_periods>
  <resources>CapitalFunds OandMFunds</resources>
  <options index='investments'>
    PlanA PlanB DoNothing;
    PlanA PlanB PlanC;
    ...
    PlanA PlanB PlanC DoNothing
  </options>
</Sets>
```

2.2 Parameters

This subsection contains information regarding the XML nodes used to define the **<Parameters>** of the optimization model being performed through LOGOS:

- **<net_present_values>**, ***comma/space-separated string, required***, specifies the NPVs for capital projects or project-option pairs. This node accepts the following optional attribute:
  - **index**, ***comma-separated string, optional***, specifies the indices of this parameter; keywords should be predefined in **<Sets>**. Valid keywords are **'investments'** and **'options'**.
    Default: investments
- **<costs>**, ***comma/space-separated string, required***, specifies the costs for capital projects or project-option pairs. This node accepts the following optional attribute:
  - **index**, ***comma-separated string, optional***, specifies the indices of this parameter; keywords should be predefined in **<Sets>**. Valid keywords are **'investments'**, **'investments, time_periods'**, **'options'**, **'options, resources'**, **'options, time_periods'**, and **'options, resources, time_periods'**.

    Default: 'investments'
- **<available_capitals>**, ***comma/space-separated string, required***, specifies the available capitals for capital projects or project-option pairs. This node accepts the following optional attribute:
  - **index**, ***comma-separated string, optional***, specifies the indices of this parameter; keywords should be predefined in **<Sets>**. Valid keywords are **'resources'**, **'time_periods'**, **'capitals'**, **'resources, time_periods'**, and **'capitals, time_periods'**
    Default: None

Example XML:

```
<Parameters>
  <net_present_values index='options'>
    27.98 27.17 0.
    -10.07 -9.78 -9.22
    ...
    8.26 7.56 7.34 0.
  </net_present_values>
  <costs index='options,resources,time_periods'>
    12.99 1.3 0 0 0
    ...
    0.01 0 0 0 0
  </costs>
  <available_capitals index="resources,time_periods">
    22.6 36.7 20.6 23.6 22.7
    0.08 0.17 0.05 0.15 0.14
```

```
  </available_capitals>
</Parameters>
```

2.3 Uncertainties

This subsection contains information regarding the XML nodes used to define the **<Uncertainties>** of the optimization model being performed through LOGOS:

- **<available_capitals>**, ***optional***, specifies the scenarios associated with available capitals. This node accepts the attribute **index**, which should be consistent with the **<available capitals>** defined in **<Parameters>**. This node accepts the following sub-nodes:
  - **<totalScenarios>**, ***integer, required***, specifies the total number of scenarios for this parameter.
  - **<probabilities>**, ***comma/space-separated float, required***, specifies the probability for each scenario. The length should be equal to the total number of scenarios.
  - **<scenarios>**, ***comma/space-separated float, required***, specifies all scenarios for this parameter. The length should be equal to the total number of scenarios multiplied by the length of this parameter, as defined in **<Parameters>**.
- **<net_present_values>**, ***optional***, specifies the scenarios associated with net_present_values. This node accepts the attribute **index**, which should be consistent with the **<net_present_values>** defined in **<Parameters>**.
  - **<totalScenarios>**, ***integer, required***, specifies the total number of scenarios for this parameter.
  - **<probabilities>**, ***comma/space-separated float, required***, specifies the probability for each scenario. The length should be equal to the total number of scenarios.
  - **<scenarios>**, ***comma/space-separated float, required***, specifies all scenarios for this parameter. The length should be equal to the total number of scenarios multiplied by the length of this parameter, as defined in **<Parameters>**.

The overall number of scenarios is the total number of scenarios for **<available_capitals>** multiplied by the total number of scenarios for **<net_present_values>**.

Example XML:

```
<Uncertainties>
  <available_capitals index="resources,time_periods">
    <totalScenarios>10</totalScenarios>
```

```
    <probabilities>
      0.5, 0.5
    </probabilities>
    <scenarios>
      20.0 34.0 17.0 20.0 18.0 0.08 0.17 0.05 0.15 0.14
      23.0 38.0 22.0 25.0 24.0 0.08 0.17 0.05 0.15 0.14
    </scenarios>
  </available_capitals>
  <net_present_values index='options'>
    <totalScenarios>9</totalScenarios>
    <probabilities>
      0.3 0.8
    </probabilities>
    <scenarios>
      13.3129 12.0228 0.0 -10.07
      ...
    </scenarios>
  </net_present_values>
</Uncertainties>
```

2.4 External Constraints

This subsection contains information regarding the XML nodes used to define the **<External Constraints>** of the optimization model being performed through LOGOS. This node accepts the following sub-node(s):

- **<constraint>**, ***string, required***, specifies the external Python module file name along with its absolute or relative path. This external Python module contains the user-defined additional constraint. **Note:** If a relative path is specified, the code first checks relative to the working directory, then it checks with respect to the location of the input file. The working directory can be specified in **<Settings>** (see Section 2.5). In addition, the extension '.py' is optional for the module file name that was inputted in this node. This sub-node also requires the following attribute:
  - **name**, ***string, required***, specifies the name of the constraint that will be added to the optimization problem.

Example XML:

```
<ExternalConstraints>
  <constraint name="con_I">externalConst</constraint>
```

```
  <constraint name="con_II">externalConstII.py</constraint>
</ExternalConstraints>
```

These constraints are Python modules, with a format automatically interpretable by LOGOS. For example, users can define their own constraint, and the code will be embedded and use the constraint as though it were an active part of the code itself. The following provides an example of a user-defined external constraint:

Example Python Function:

```
# External constraint function
import numpy as np
import pyomo.environ as pyomo

def initialize():
  """
    Optional Method
    Optimization model parameters values can be updated/modified
    without directly accessing the optimization model.
    Value(s) will be updated in-place.
    @ In, None
    @ Out, updateDict, dict, {paramName:paramInfoDict},  where
      paramInfoDict contains {Indices:Values}
      Indices are parameter indices (either strings or tuples of
      strings, depending on whether there is one or
      more than one dimension). Values are the new values being
      assigned to the parameter at the given indices.
  """
  updateDict = {'available_capitals':{'None':16},
                'costs':{'1':1,'2':3,'3':7,'4':4,'5':8,
                         '6':9,'7':6,'8':10,'9':2,'10':5}
               }
  return updateDict

def constraint(var, sets, params):
  """
    Required Method
    External constraint provided by users that will be added to
    optimization problem
    @ In, sets, dict, all "Sets" provided in the Logos input
      file will be stored and available in this dictionary,
      i.e. {setName: setObject}
    @ In, params, dict, all "Parameters" provided in the
```

```
    Logos input file will be stored and
    available in this dictionary, i.e. {paramName: paramObject}
  @ In, var, object, the internally used decision variable,
    the dimensions/indices of this variable depend the type of
    optimization problems (i.e. "<problem_type>" from Logos
    input file). Currently, we will accept the following
    problem types:

    1. "singleknapsack": in this case, "var" will be var[:],
       where the index will be the element from
       xml node of "investment" in Logos input file.

    2. "multipleknapsack": in this case, "var" will be var[:,:],
       where the indices are the combinations element from set
       "investment" and element from set "capitals" in Logos
       input file

    3. "mckp": in this case, "var" will be var[:,:], where the
       indices are the combinations element from set
       "investment" and element from set "options" in Logos
       input file

    (Note that any element that is used as index will be
    converted to a string even if a number is provided in
    the Logos input file).

  @ Out, constraint, tuple, either (constraintRule,)
    or (constraintRule, indices)

  (Note that any modifications in provided sets and params
  will only have impact on this local module,
  i.e. the external constraint. In other words, the Sets
  and Params used in the internal constraints and
  objective will be kept unchanged!)
"""
# All sets and parameters can be retrieved from dictionary
# "sets" and "params" investments = sets['investments']

def constraintRule(self, i):
  """
    Expression for user provided external constraint
    @ In, self, object, required to present, but not used
```

```
      @ In, i, str, element for the index set
      @ Out, constraintRule, function expression, expression
        to define user provided constraint

      Note that: Constraints can be indexed by lists or sets.
      When the return of function "constraint" contains
      lists or sets except the "constraintRule", the elements
      are iteratively passed to the rule function. If there
      is more than one, then the cross product is sent.
      For example, this constraint could be interpreted as
      placing limit on "ith" decision variable "var".
      A list of constraints for all "ith" decision variable
      "var" will be added to the optimization model
    """
    return var[i] <= 1

  # A tuple is required for the return, the first element
  # should be always the "constraintRule",
  # while the rest of elements are the lists or sets
  # if the user wants to construct the constraints
  # iteratively (See the docstring in "constraintRule"),
  # otherwise, keep it empty
  return (constraintRule, investments)
```

2.5 Settings: Options for Optimization

This subsection contains information regarding the XML nodes used to define the **<Settings>** of the optimization model being performed through LOGOS:

- **<problem_type>**, ***string, required parameter***, specifies the type of optimization problem. Available types include **'SingleKnapsack'**, **'MultipleKnapsack'**, and **'MCKP'** for risk-informed stochastic optimization. Available types include **'droskp'**, **'dromkp'**, and **'dromckp'** for distributionally robust optimization. Available types include **'cvarskp'**, **'cvarmkp'**, and **'cvarmckp'**.
- **<solver>**, ***string, optional parameter***, represents available solvers including **<cbc>** from `https://github.com/coin-or/Cbc.git` and **<glpk>** from `https://www.gnu.org/software/glpk/`.
- **<sense>**, ***string, optional parameter***, specifies **'minimize'** or **'maximize'** for minimization or maximization, respectively.

Default: minimize

- **<mandatory>**, ***comma/space-separated string, optional parameter***, specifies regulatorily mandated or must-do projects.
- **<nonSelection>**, ***boolean, optional parameter***, indicates whether the investments options includes *DoNothing* option.
  Default: False
- **<lowerBounds>**, ***comma/space-separated integers, optional parameter***, specifies the lower bounds for decision variables.
- **<upperBounds>**, ***comma/space-separated integers, optional parameter***, specifies the upper bounds for decision variables.
- **<consistentConstraintI>**, ***string, optional parameter***, indicates whether this constraint is enabled.
  Default: True
- **<consistentConstraintII>**, ***string, optional parameter***, indicates whether this constraint is enabled or not.
  Default: False
- **<solverOptions>**, ***optional parameter***, accepts different options for the given solver provided in **<solver>**. A simple XML node only containing node tags and node texts can be used to provide the options for the solver. For example:

```
<solverOptions>
  <threads>1</threads>
  <StochSolver>EF</StochSolver>
</solverOptions>
```

  In addition, if the problem type is distributionally robust optimization, additional option **<radius_ambiguity>** can be used to control the Wasserstein distance. See Section 5. If the problem type is conditional value at risk (See Section 6), additional options are available:

  - **<risk_aversion>**, ***float within*** $[0, 1]$***, optional parameter***, indicates the weight on maximizing expected NPV versus penalizing solutions that yield low-NPV scenarios.
  - **<confidence_level>**, ***float within*** $[0, 1]$***, optional parameter***, indicates the confidence level, i.e. the α-percentile of the loss.

Example XML:

```
<Settings>
  <mandatory>
    PresurizerReplacement
```

```
    ...
    ReplaceInstrumentationAndControlCables
  </mandatory>
  <nonSelection>True</nonSelection>
  <consistentConstraintI>True</consistentConstraintI>
  <consistentConstraintII>True</consistentConstraintII>
  <solver>cbc</solver>
  <solverOptions>
    <threads>1</threads>
    <StochSolver>EF</StochSolver>
  </solverOptions>
  <sense>maximize</sense>
  <problem_type>mckp</problem_type>
</Settings>
```

3 Deterministic Capital Budgeting

We consider a capital budgeting problem for a nuclear generation station, with possible extension to a larger fleet of plants. Due to limited resources, we can only select a subset from a list of several candidate capital projects. Our goal is to maximize overall NPV associated with the selected subset. In doing so, we must respect resource limits and capture key structural and stochastic dependencies of the system, although in this section we start with the simpler deterministic case, ignoring randomness. Example projects include upgrading a steam turbine, refurbishing or replacing a set of reactor coolant pumps, and replacing a set of feed-water heaters.

Indexes and sets:

$t \in T$	time periods (years)
$i \in I$	investment candidate projects
$j \in J_i$	options for selecting project i
$k \in K$	types of resources

Data:

a_{ij}	reward (revenue minus financial cost) of selecting project i via option j
b_{kt}	available budget for a resource of type k in year t
c_{ijkt}	consumption of resource of type k in year t if project i is performed via option j

Decision variables:

x_{ij}	1 if project i is selected via option j; 0 otherwise

Formulation:

$$\max_{x} \quad \sum_{i \in I, j \in J_i} a_{ij} x_{ij} \tag{1a}$$

$$s.t. \quad \sum_{j \in J_i} x_{ij} = 1, i \in I \tag{1b}$$

$$\sum_{i \in I, j \in J_i} c_{ijkt} x_{ij} \leq b_{kt}, k \in K, t \in T \tag{1c}$$

$$x_{ij} \in \{0, 1\}, j \in J_i, i \in I. \tag{1d}$$

The decision variables, x_{ij}, indicate whether we choose to do project i by means j. Restated, if $x_{ij} = 1$, then we recommend doing project i via option j; taken together, these decision variables produce both a portfolio of selected projects and a schedule for performing those projects over

time. The set of available options, $j \in J_i$, can explicitly include the "do-nothing" option, and the first constraint ensures that we choose exactly one option from the available set for each project, including the possibility of selecting the do-nothing option. Even if we select the do-nothing option for a project, it induces an NPV, a_{ij}, which may be negative, representing growing O&M costs, losses in plant efficiency, etc. The second structural constraint ensures that the budget of each resource k is respected in each year t. The objective function includes the NPV for each project-option pair, a_{ij}, and the correct NPV is selected by the $0-1$ decision variable, x_{ij}.

We note that certain projects must be done (e.g., for safety and or regulatory reasons). This can be handled within the mathematical formulation just given, without introducing additional constructs. The set J_i typically includes a do-nothing option for each project, but when project i must be done, we simply do not include the do-nothing option. Mathematically, one alternative is to disclude an explicit do-nothing option, replace the first structural constraint with an inequality, and add an additional set of must-do projects with an equality constraint. Both options are mathematically equivalent and simply represent a choice to be made by the analyst. In LOGOS, we use **<mandatory>** and **<nonSelection>** to handle these conditions. **<mandatory>** is used to specify the must-do projects, while **<nonSelection>** is used to activate the do-nothing option.

Note: If a project is listed under **<mandatory>**, the do-nothing option is not allowed for this project. In addition, handling the do-nothing option implicitly leads to the NPV being calculated relative to that of the do-nothing option.

The objective of capital budgeting is to find the right combination of binary decisions for every investment so that overall profit is maximized. The output is a collection of projects to be carried out, and we refer to this selected collection of projects as a "project portfolio". However, as is frequently the case for capital budgeting with NPP applications, in practice, several optional constraints, such as resources/liabilities, dependencies/synergies, options, time windows for every investment, etc., have to be fulfilled. This leads to a various variations of the knapsack problem. In the following subsection, we will present several different variants of the above capital budgeting problem (i.e., variants of the knapsack problem).

3.1 Single Knapsack Problem Optimization

3.1.1 Simple Knapsack Problem

The simple knapsack problem (KP) for capital budgeting can be defined as follows: we are given an instance of the capital budgeting problem with investment set I, consisting of I investments i with profit a_i (e.g. NPV), cost c_i, and available budget b. The objective is to select a subset of I such that the total profit of the selected investments is maximized and the total cost does not exceed b. Alternatively, KP can be formulated as a solution of the following linear integer programming formulation:

$$\max_{x} \quad \sum_{i \in I} a_i x_i \tag{2a}$$

$$s.t. \quad \sum_{i \in I} c_i x_i \leq b \tag{2b}$$

$$x_i \in \{0, 1\}, i \in I. \tag{2c}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10
    </investments>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      18,20,17,19,25,21,27,23,25,24
    </net_present_values>
    <costs index="investments">
      1,3,7,4,8,9,6,10,2,5
    </costs>
    <available_capitals>
      15
    </available_capitals>
  </Parameters>

  <Settings>
    <solver>cbc</solver>
    <sense>maximize</sense>
  </Settings>
</Logos>
```

When running this case, LOGOS would generate a CSV (comma separated values) file containing solutions for the optimization problem (i.e. values of decision variables and maximum profit [MaxNPV is used to describe the maximum profit]). The header of this CSV file contains the indices listed under **<investments>** used as indices for decision variables and the objective variable **MaxNPV**. The data provides the values for both decision variables and the objective variable.

Example LOGOS output CSV:

```
1,2,3,4,5,6,7,8,9,10,MaxNPV
1.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,1.0,106.0
```

In this case, projects **1, 2, 4, 9, and 10** are selected with a maximum profit of 106.0.

3.1.2 Bounded Knapsack Problem

In the capital budgeting problem described above, it may be the case that not all investments/projects are different from each other. For example, in practice, there may be given a number (n_i) of identical pumps/valves to be replaced. In this case, the number of decision variables is equal to the number of different investments, rather than the total number of investments. The constraint for the decision variables becomes:

$$0 \leq x_i \leq n_i, i \in N \tag{3}$$

The resulting problem is called the bounded knapsack problem (BKP) and is formally defined as:

$$\max_{x} \quad \sum_{i \in I} a_i x_i \tag{4a}$$

$$s.t. \quad \sum_{i \in I} c_i x_i \leq b \tag{4b}$$

$$x_i \in \{0, n_i\}, i \in I. \tag{4c}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
      14, 15, 16, 17, 18, 19, 20, 21, 22
    </investments>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      150,35,200,60,60,45,60,40,30,10,70,
      30,15,10,40,70,75,80,20,12,50,10
    </net_present_values>
```

```
    <costs index="investments">
      9,13,153,50,15,68,27,39,23,52,11,32,
      24,48,73,42,43,22,7,18,4,30
    </costs>
    <available_capitals>
      400
    </available_capitals>
  </Parameters>

  <Settings>
    <lowerBounds>
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    </lowerBounds>
    <upperBounds>
      1,1,2,2,2,3,3,3,1,3,1,
      1,2,2,1,1,1,1,1,2,1,2
    </upperBounds>
    <solver>glpk</solver>
    <sense>maximize</sense>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
1,2,...,21,22,MaxNPV
1.0,1.0,...,1.0,0.0,1010.0
```

3.1.3 Multi-Dimensional Knapsack Problem: DKP

Moving in a different direction, we now take into account not only the cost constraint, but also the limited commitment of critical resources, including: (i) capital cost, (ii) O&M costs, (iii) time and labor-hours during a planned outage, and (iv) personnel, installation and maintenance equipment, workspaces, etc.. Denoting the cost of every investment by c_{ik} for each resource k and introducing the corresponding limited resource b_k, we can formulate the capital budgeting problem as a multi-dimensional or D-dimensional knapsack problem formally defined by:

$$\max_{x} \quad \sum_{i \in I} a_i x_i \tag{5a}$$

$$s.t. \quad \sum_{i \in I} c_{ik} x_i \leq b_k, k \in K \tag{5b}$$

$$x_i \in \{0, 1\}, i \in I. \tag{5c}$$

Where the limited resources set is denoted by K, consisting of k "colors" of money within capital costs, O&M costs, personnel availability, etc. Another example is provided if the plant has multi-year investments. Consider a DKP problem in which the costs of each investment and the available capitals vary according to time period t. By defining c_{it} as the cost of investment i at time period i, and b_t as the available capital at time period t, we get:

$$\max_{x} \quad \sum_{i \in I} a_i x_i \tag{6a}$$

$$s.t. \quad \sum_{i \in I} c_{it} x_i \leq b_t, t \in T \tag{6b}$$

$$x_i \in \{0, 1\}, i \in I. \tag{6c}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9
    </investments>
    <time_periods>
      1,2,3,4,5
    </time_periods>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      2.315,0.824,22.459,60.589,0.667,5.173,4.003,0.582,0.122
    </net_present_values>
    <costs index="investments,␣time_periods">
```

```
      0.219,0.257,0.085,0.0,0.0,
      0.0,0.0,0.122,0.103,0.013,
      5.044,1.839,0.0,0.0,0.0,
      6.74,6.134,10.442,0.0,0.0,
      0.425,0.0,0.0,0.0,0.0,
      2.125,2.122,0.0,0.0,0.0,
      2.387,0.19,0.012,2.383,0.192,
      0.0,0.95,0.0,0.0,0.0,
      0.03,0.03,0.688,0.0,0.0
    </costs>
    <available_capitals index="time_periods">
      0.665,4.712,9.642,3.458,1.683
    </available_capitals>
  </Parameters>

  <Settings>
    <solver>glpk</solver>
    <sense>maximize</sense>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
1,2,3,4,5,6,7,8,9,MaxNPV
1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,4.388
```

3.2 Multiple Knapsack Problem Optimization

Another interesting variant of the capital budgeting problem arises if we consider maintenance for multiple units in a NPP in parallel, i.e. it has to be decided whether to accept a particular replacement and, in the positive case, in which unit to conduct the corresponding replacement. This can be formulated by introducing a binary decision variable for every maintenance-unit combination. If there are I investments (investment set I) on the list of maintenance requests and m units (unit set M) available, we use binary variables:

$$x_{im} \in \{0, 1\}, i \in I, m \in M \tag{7}$$

The resulting problem is called the multiple knapsack problem (MKP), and the mathematical formulation is given by:

$$\max_{x} \quad \sum_{m \in M} \sum_{i \in I} a_i x_{im} \tag{8a}$$

$$s.t. \quad \sum_{i \in I} c_i x_{im} \leq b_m, m \in M \tag{8b}$$

$$\sum_{m \in M} x_{im} \leq 1 \tag{8c}$$

$$x_{im} \in \{0, 1\}, i \in I, m \in M. \tag{8d}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10
    </investments>
    <capitals>
      unit_1, unit_2
    </capitals>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      78, 35, 89, 36, 94, 75, 74, 79, 80, 16
    </net_present_values>
    <costs index="investments">
      18, 9, 23, 20, 59, 61, 70, 75, 76, 30
    </costs>
    <available_capitals index="capitals">
      103, 156
    </available_capitals>
  </Parameters>

  <Settings>
    <solver>cbc</solver>
    <sense>maximize</sense>
    <problem_type>MultipleKnapsack</problem_type>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
1,2,3,4,5,6,7,8,9,10,capitals,MaxNPV
0.0,0.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,unit_1,452.0
1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,unit_2,452.0
```

3.3 Multiple-Choice Multi-Dimensional Knapsack Problem Optimization

Another quite different variant of the capital budgeting problem appears if there may be multiple ways to carry out each investment/project. Each investment i, however, exists in a number of options in which the j-th option has cost c_{ij} and profit value a_{ij}. This problem can be expressed as the multiple-choice knapsack problem (MCKP). Assume J_i is the set of different options for investment i. Using the decision variables x_{ij} to denote whether option j was chosen from the set J_i, the mathematical formulation of MCKP is given by:

$$\max_{x} \quad \sum_{i \in I, j \in J_i} a_{ij} x_{ij} \tag{9a}$$

$$s.t. \quad \sum_{j \in J_i} x_{ij} = 1, i \in I \tag{9b}$$

$$\sum_{i \in I, j \in J_i} c_{ij} x_{ij} \leq b \tag{9c}$$

$$x_{ij} \in \{0, 1\}, j \in J_i, i \in I. \tag{9d}$$

Considering the limited resources and multi-year investments, the MCKP may be extended to a D-dimensional MCKP problem (D-MCKP). For example, a project may be performed over a three-year period–say, years 't, $t+1$, $t+2$'–or the start of the project could instead be two years hence, changing the equation to '$t+2$, $t+3$, $t+4$'. Alternatively, at increased cost and benefit, it may be possible to complete the project in two years: 't, $t+1$', or '$t+2$, $t+3$'. When selecting a project to uprate plant capacity, we may have the options of increasing it by 3% or 6%. In these cases, the problem can be expressed as the D-MCKP. This problem is formally defined as follows:

$$\max_{x} \quad \sum_{i \in I, j \in J_i} a_{ij} x_{ij} \tag{10a}$$

$$s.t. \quad \sum_{j \in J_i} x_{ij} = 1, i \in I \tag{10b}$$

$$\sum_{i \in I, j \in J_i} c_{ijt} x_{ij} \leq b_t, t \in T \tag{10c}$$

$$x_{ij} \in \{0, 1\}, j \in J_i, i \in I. \tag{10d}$$

or in the same as the problem described at the beginning of this chapter (see Section 3):

$$\max_{x} \quad \sum_{i \in I, j \in J_i} a_{ij} x_{ij} \tag{11a}$$

$$s.t. \quad \sum_{j \in J_i} x_{ij} = 1, i \in I \tag{11b}$$

$$\sum_{i \in I, j \in J_i} c_{ijkt} x_{ij} \leq b_{kt}, k \in K, t \in T \tag{11c}$$

$$x_{ij} \in \{0, 1\}, j \in J_i, i \in I. \tag{11d}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
    </investments>
    <options index='investments'>
      1;
      1;
      1;
      1,2,3;
      1,2,3,4;
      1,2,3,4,5,6,7;
      1;
      1;
```

```
      1;
      1;
      1;
      1;
      1;
      1;
      1;
      1;
      1
    </options>
  </Sets>

  <Parameters>
    <net_present_values index='options'>
      2.046 2.679 2.489 2.61 2.313 1.02 3.013 2.55 3.351 3.423
         3.781 2.525
      2.169 2.267 2.747 4.309 6.452 2.849 7.945 2.538 1.761
         3.002 3.449
      2.865 3.999 2.283 0.9 8.608
    </net_present_values>
    <costs index='options'>
      36538462
      83849038
      4615385
      2788461538
      2692307692
      5480769231
      1634615385
      2981730768
      7211538462
      9038461538
      649038462
      650000000
      216346154
      212500000
      3076923077
      3942307692
      1144230769
      675721154
      1442307692
      99711538
      4807692
```

```
      123076923
      138461538
      86538462
      108653846
      75092404
      6413462
      147932692
    </costs>
    <available_capitals>
      15E9
    </available_capitals>
  </Parameters>

  <Settings>
    <solver>cbc</solver>
    <sense>maximize</sense>
    <problem_type>mckp</problem_type>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
1__1,2__1,3__1,4__1,4__2,4__3,...,17__1,MaxNPV
1.0,1.0,1.0,1.0,0.0,0.0,...,1.0,59.82600000000001
```

In the output file, the names **"investmentsIndex"**__**"optionsIndex"** are used to specify the decision variable. For example, **1**__**1** indicates that investment **1** with option **1** is selected.

4 Prioritizing Project Selection to Hedge Against Uncertainty

One limitation of traditional optimization models for capital budgeting is that they do not account for risk/uncertainty in profit and cost streams associated with individual projects, nor do they account for risk in resource availability in future years. Projects can incur cost overruns, especially when projects are large, performed infrequently, or when there is risk regarding technical viability, external contractors, and/or suppliers of requisite parts and materials. Occasionally, projects are performed ahead of schedule and with savings in cost. Planned budgets for capital improvements can be cut, and key personnel may be lost. Or, there may be surprise budgetary windfalls for maintenance activities due to decreased costs for "unplanned" maintenance. In such cases, how should we resolve capital budgeting when we have risk forecasts for costs, profits, and budgets? One approach is to re-solve the models described in the previous section once refined forecasts for these parameters become available. However, it is not always practical to fully revise a project portfolio whenever better forecasts become available.

In order to prioritize the project selection using a risk forecast for these parameters, the two-stage stochastic optimization model [3] is employed to provide priority lists to decision-makers to support better risk-informed decisions. Its inputs include those described in above sections for different variants of the capital budgeting problem, except that a probabilistic description of the uncertain parameters is integrated into the optimization process. The two-stage stochastic optimization model forms a priority list as its first-stage decision, then forms a corresponding project portfolio for each scenario as its second-stage decision. When forming the optimal second-stage project portfolio under a specific scenario, the stochastic optimization model ensures that the portfolio is consistent with the first-stage prioritization (i.e., a project can be selected only if all high-priority projects are also selected.) Thus, the portfolios of projects corresponding to different scenarios are nested.

The notation and formulation of the risk-informed models are as follows:

Indices and sets:

$t \in T$	time periods (years)
$i, i^{'} \in I$	candidate projects
$j \in J_i$	options for selecting project i
$k \in K$	types of resources
$m \in M$	units of NPP
$\omega \in \Omega$	scenarios

Data:

a_i^ω	reward for selecting project i under scenario ω
a_{ij}^ω	reward for selecting project i via option j under scenario ω
b^ω	available budget under scenario ω
b_k^ω	available budget for a resource of type k under scenario ω
b_t^ω	available budget in year t under scenario ω
b_m^ω	available budget for unit m under scenario ω
b_{kt}^ω	available budget for a resource of type k in year t under scenario ω
c_i^ω	cost of investment i under scenario ω
c_{ik}^ω	consumption of resource of type k if project i is selected under scenario ω
c_{ijt}^ω	consumption of resource in year t if project i is performed via option j under scenario ω
c_{ijkt}^ω	consumption of resource of type k in year t if project i is performed via option j under scenario ω
q^ω	probability of scenario ω

Decision variables:

x_i^ω	1 if project i is selected under scenario ω; 0 otherwise
x_{im}^ω	1 if project i is selected for unit m under scenario ω; 0 otherwise
x_{ij}^ω	1 if project i is selected via option j under scenario ω; 0 otherwise
$y_{ii'}$	1 if project i has no lower priority than project $i^{'}$; 0 otherwise

4.1 Risk-Informed Single Knapsack Problem Optimization

4.1.1 Risk-Informed Simple Knapsack Problem

Formulation:

$$\max_{x} \quad \sum_{\omega \in \Omega} q^{\omega} \sum_{i \in I} a_i^{\omega} x_i^{\omega} \tag{12a}$$

$$s.t. \quad \sum_{i \in I} c_i^{\omega} x_i^{\omega} \leq b^{\omega} \tag{12b}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{12c}$$

$$x_i^{\omega} \geq x_{i'}^{\omega} + y_{ii'} - 1, i \neq i' \tag{12d}$$

For simplicity in what follows, the variable $y_{ii'} = 1$ means that project i is of higher priority than i', even though the variable definition allows for ties (i.e., projects of the same priority). Constraint (12b) requires us to be within budget under each scenario. Constraint (12c) indicates that either project i is of higher priority than project i', or vice versa, or that both are of equal priority (i.e., a tie). Constraint (12d) indicates that if project i is of higher priority than project i' ($y_{ii'} = 1$), and we select the lower priority project, then we must also select the higher priority project; if $y_{ii'} = 0$, or if $x_{i'}^{\omega} = 0$, then the constraint is vacuous. In order to handle the risk in the capital budgeting problems, the entity **<Uncertainties>** (see section 2.3) is used to specify different scenarios of input parameters.

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10
    </investments>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      18,20,17,19,25,21,27,23,25,24
    </net_present_values>
    <costs index="investments">
      1,3,7,4,8,9,6,10,2,5
    </costs>
    <available_capitals>
      15
```

```
    </available_capitals>
  </Parameters>

  <Uncertainties>
    <available_capitals>
      <totalScenarios>10</totalScenarios>
      <probabilities>
        0.012, 0.019, 0.032, 0.052, 0.086, 0.142, 0.235, 0.188,
           0.141, 0.093
      </probabilities>
      <scenarios>
        11, 12, 13, 14, 15, 16, 17, 18, 19, 20
      </scenarios>
    </available_capitals>
    <net_present_values>
      <totalScenarios>2</totalScenarios>
      <probabilities>
        0.3, 0.7
      </probabilities>
      <scenarios>
        18,20,17,19,25,21,27,23,25,24,
        18,20,17,19,25,21,27,23,25,24
      </scenarios>
    </net_present_values>
  </Uncertainties>
  ...
</Logos>
```

When running this case, LOGOS would generate a CSV (comma separated values) file containing solutions for the optimization problem (i.e. values of decision variables and maximum profit [MaxNPV is used to describe the maximum profit]). The header of this CSV file contains the indices listed under **<investments>** used as indices for decision variable, the objective variable **MaxNPV**, the scenario name and the associated probability weight. The data provides the values for both decision variables and the objective variable.

Example LOGOS output CSV:

```
1,10,2,3,4,5,6,7,8,9,ScenarioName,ProbabilityWeight,MaxNPV
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_1,0.0036,70.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_6,0.0224,70.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_5,0.0096,70.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_4,0.0133,70.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_3,0.0057,70.0
1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_2,0.0084,70.0
```

```
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_7,0.0156,94.0
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_8,0.0364,94.0
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_9,0.0258,94.0
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_12,0.0994,94.0
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_11,0.0426,94.0
1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_10,0.0602,94.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_16,0.1316,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_19,0.0279,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_15,0.0564,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_20,0.0651,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_14,0.1645,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_13,0.0705,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_17,0.0423,114.0
1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,scenario_18,0.0987,114.0
```

4.1.2 Risk-Informed Multi-Dimensional Knapsack Problem

Formulation:

$$\max_{x} \quad \sum_{\omega \in \Omega} q^{\omega} \sum_{i \in I} a_i^{\omega} x_i^{\omega} \tag{13a}$$

$$s.t. \quad \sum_{i \in I} c_{it}^{\omega} x_i^{\omega} \leq b_t^{\omega}, t \in T \tag{13b}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{13c}$$

$$x_i^{\omega} \geq x_{i'}^{\omega} + y_{ii'} - 1, i \neq i' \tag{13d}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
    </investments>
    <time_periods>
      1,2,3,4,5
    </time_periods>
  </Sets>

  <Parameters>
```

```
  <net_present_values index="investments">
    2.315,0.824,22.459,60.589,0.667,5.173,4.003,0.582,0.122,
    -2.870,-0.102,-0.278,-0.322,-3.996,-0.246,-20.155
  </net_present_values>
  <costs index="investments, time_periods">
    0.219,0.257,0.085,0.0,0.0,
    0.0,0.0,0.122,0.103,0.013,
    5.044,1.839,0.0,0.0,0.0,
    6.74,6.134,10.442,0.0,0.0,
    0.425,0.0,0.0,0.0,0.0,
    2.125,2.122,0.0,0.0,0.0,
    2.387,0.19,0.012,2.383,0.192,
    0.0,0.95,0.0,0.0,0.0,
    0.03,0.03,0.688,0.0,0.0,
    0,0.2,0.763,0.739,2.539,
    0.081,0.032,0,0,0,
    0.3,0,0,0,0,
    0.347,0,0,0,0,
    4.025,0.297,0,0,0,
    0.095,0.095,0.095,0,0,
    5.487,5.664,0.5,6.803,6.778
  </costs>
  <available_capitals index="time_periods">
    18,18,18,18,18
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
    <totalScenarios>10</totalScenarios>
    <probabilities>
      0.012, 0.019, 0.032, 0.052, 0.086, 0.142, 0.235, 0.188,
        0.141, 0.093
    </probabilities>
    <!--
      scenarios is ordered by numberScenarios *
        parametersIndex, the number of
      scenarios is determined by the number of elements in
        <probabilities>,
      for this case total element in scenarios:
      numberScenarios * time_periods = 10 * 5
    -->
```

```
      <scenarios>
        11, 11, 11, 11, 11,
        12, 12, 12, 12, 12,
        13, 13, 13, 13, 13,
        14, 14, 14, 14, 14,
        15, 15, 15, 15, 15,
        16, 16, 16, 16, 16,
        17, 17, 17, 17, 17,
        18, 18, 18, 18, 18,
        19, 19, 19, 19, 19,
        20, 20, 20, 20, 20
      </scenarios>
    </available_capitals>
  </Uncertainties>
  <Settings>
    <mandatory>10,11,12,13,14,15,16</mandatory>
    <solver>cbc</solver>
    <sense>maximize</sense>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
1,10,11,12,13,14,15,16,2,3,4,5,6,7,8,9,ScenarioName,ProbabilityWeight,MaxNPV
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,scenario_1,0.012,-23.581
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_2,0.019,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_3,0.032,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_4,0.052,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_5,0.086,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_6,0.142,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,scenario_7,0.235,-23.459
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0,0.0,0.0,1.0,1.0,scenario_8,0.188,37.130
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0,0.0,0.0,1.0,1.0,scenario_9,0.141,37.1230
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0,1.0,0.0,1.0,1.0,scenario_10,0.093,42.303
```

4.2 Risk-Informed Multiple Knapsack Problem Optimization

Model formulation:

$$\max_{x,y} \sum_{\omega \in \Omega} q^{\omega} \sum_{i \in I} \sum_{m \in M} a_i^{\omega} x_{im}^{\omega} \tag{14}$$

$$s.t. \quad y_{ii'} + y_{i'i} \geq 1, \; i < i', \mathrm{i}, i' \in I \tag{15}$$

$$\sum_{m=1}^{M} x_{im}^{\omega} \geq \sum_{m=1}^{M} x_{i'm}^{\omega} + y_{ii'} - 1,\ i \neq i', \mathrm{i}, i' \in I, \omega \in \Omega \tag{16}$$

Constraint (15) indicates that either project i is of higher priority than project $i^{'}$, or vice versa, or that both are of equal priority (i.e., a tie). Constraint (16) indicates that if project i is higher priority than project $i^{'}$ ($y_{ii'} = 1$), and we select the lower priority project *for some unit*, then we must also select the higher priority project; if $y_{ii'} = 0$, or if $\sum_{m=1}^{M} x_{i'm}^{\omega} = 0$ then the constraint is vacuous.

$$\sum_{i \in I} c_i^{\omega} x_{im}^{\omega} \ \leq b_m^{\omega},\ m \in M, \omega \in \Omega \tag{17}$$

Constraint (17) requires that we be within budget for each unit under each scenario.

$$\sum_{m \in M} x_{im}^{\omega} \leq 1,\ i \in I, \omega \in \Omega \tag{18}$$

Constraint (18) simultaneously ensures that we select project i only for one unit.

Example LOGOS input XML:

```
<Sets>
  <investments>
    1,2,3,4,5,6,7,8,9,10
  </investments>
  <capitals>
    unit_1, unit_2
  </capitals>
</Sets>

<Parameters>
  <net_present_values index="investments">
    78, 35, 89, 36, 94, 75, 74, 79, 80, 16
  </net_present_values>
  <costs index="investments">
    18, 9, 23, 20, 59, 61, 70, 75, 76, 30
  </costs>
  <available_capitals index="capitals">
    103, 156
  </available_capitals>
```

```
  </Parameters>

  <Uncertainties>
    <available_capitals>
      <totalScenarios>2</totalScenarios>
      <probabilities>
        0.3 0.7
      </probabilities>
      <scenarios>
        103, 156,
        103, 156
      </scenarios>
    </available_capitals>
    <net_present_values>
      <totalScenarios>2</totalScenarios>
      <probabilities>
        0.3, 0.7
      </probabilities>
      <scenarios>
        78, 35, 89, 36, 94, 75, 74, 79, 80, 16,
        78, 35, 89, 36, 94, 75, 74, 79, 80, 16
      </scenarios>
    </net_present_values>
  </Uncertainties>

  <Settings>
    <solver>glpk</solver>
    <sense>maximize</sense>
    <problem_type>MultipleKnapsack</problem_type>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
"('1', 'unit_1')","('1', 'unit_2')","('10', 'unit_1')","('10', 'unit_2')","('2', 'unit_1')","('2', 'unit_2')","('3', 'unit_1')",
"('3', 'unit_2')","('4', 'unit_1')","('4', 'unit_2')","('5', 'unit_1')","('5', 'unit_2')","('6', 'unit_1')","('6', 'unit_2')",
"('7', 'unit_1')","('7', 'unit_2')","('8', 'unit_1')","('8', 'unit_2')","('9', 'unit_1')","('9', 'unit_2')",
ScenarioName, ProbabilityWeight, MaxNPV
1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, scenario_1, 0.09, 452.0
1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, scenario_2, 0.21, 452.0
1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, scenario_3, 0.21, 452.0
0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, scenario_4, 0.49, 452.0
```

4.3 Risk-Informed Multiple-Choice Multi-Dimensional Knapsack Problem Optimization

Model formulation:

$$\max_{x,y} \sum_{\omega \in \Omega} q^{\omega} \sum_{i \in I} \sum_{j \in J_i} a_{ij}^{\omega} x_{ij}^{\omega} \tag{19}$$

$$s.t. \quad y_{ii'} + y_{i'i} \geq 1, \; i < i', \mathrm{i}, i' \in I \tag{20}$$

$$\sum_{j=1}^{J_i} x_{ij}^{\omega} \geq \sum_{j=1}^{J_i} x_{i'j}^{\omega} + y_{ii'} - 1, \; i \neq i', \mathrm{i}, i' \in I, \omega \in \Omega \tag{21}$$

Constraint (20) indicates that either project i is of higher priority than project i', or vice versa, or that both are of equal priority (i.e., a tie). Constraint (21) indicates that if project i is higher priority than project i' $y_{ii'} = 1$, and we select the lower priority project *under some option*, then we must also select the higher priority project; if $y_{ii'} = 0$, or if $\sum_{j=1}^{J_i} x_{i'j}^{\omega} = 0$, then the constraint is vacuous.

$$\sum_{i \in I} \sum_{j \in J_i} \mathrm{c}_{ijkt}^{\omega} x_{ij}^{\omega} \; \leq b_{kt}^{\omega}, \; k \in K, t \in T, \omega \in \Omega \tag{22}$$

Constraint (22) requires that we be within budget in each time period, for each resource type, and under each scenario.

$$\sum_{j \in J_i} x_{ij}^{\omega} \leq 1, \; i \in I, \omega \in \Omega \tag{23}$$

Constraint (23) simultaneously ensures that we select project i via, at most, one option. Note that this illustrates a situation in which we must include the *DoNothing* option among the alternatives to optional projects.

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
```

```
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
  </investments>
  <options index='investments'>
    1;
    1;
    1;
    1,2,3;
    1,2,3,4;
    1,2,3,4,5,6,7;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1
  </options>
</Sets>

<Parameters>
  <net_present_values index='options'>
    2.046
    2.679
    2.489
    2.61
    2.313
    1.02
    3.013
    2.55
    3.351
    3.423
    3.781
    2.525
    2.169
    2.267
    2.747
    4.309
    6.452
```

```
    2.849
    7.945
    2.538
    1.761
    3.002
    3.449
    2.865
    3.999
    2.283
    0.9
    8.608
  </net_present_values>
  <costs index='options'>
    36538462
    83849038
    4615385
    2788461538
    2692307692
    5480769231
    1634615385
    2981730768
    7211538462
    9038461538
    649038462
    650000000
    216346154
    212500000
    3076923077
    3942307692
    1144230769
    675721154
    1442307692
    99711538
    4807692
    123076923
    138461538
    86538462
    108653846
    75092404
    6413462
    147932692
  </costs>
```

```
    <available_capitals>
      15E9
    </available_capitals>
  </Parameters>

  <Uncertainties>
    <available_capitals>
      <totalScenarios>3</totalScenarios>
      <probabilities>
        0.2,0.6,0.2
      </probabilities>
      <scenarios>
        5E9,10E9,15E9
      </scenarios>
    </available_capitals>
  </Uncertainties>

  <Settings>
    <solver>cbc</solver>
    <solverOptions>
      <threads>1</threads>
      <StochSolver>EF</StochSolver>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>mckp</problem_type>
  </Settings>
</Logos>
```

Example LOGOS output CSV:

```
"('1', '1')","('10', '1')","('11', '1')","('12', '1')","('13', '1')","('14', '1')","('15', '1')","('16', '1')",
"('17', '1')","('2', '1')","('3', '1')","('4', '1')","('4', '2')","('4', '3')","('5', '1')","('5', '2')","('5', '3')",
"('5', '4')","('6', '1')","('6', '2')","('6', '3')","('6', '4')","('6', '5')","('6', '6')","('6', '7')","('7', '1')",
"('8', '1')","('9', '1')",ScenarioName,ProbabilityWeight,MaxNPV
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,scenario_1,0.2,53.865
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,scenario_2,0.6,59.488
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,scenario_3,0.2,59.826
```

5 Distributionally Robust Optimization

We consider another risk-averse decision making approach using distributionally robust optimization (DRO). To this end, we begin with a nominal stochastic optimization problem:

$$\max_{s \in S} \sum_{\sigma \in \Sigma} q^{\sigma} f(s, \xi^{\sigma}) \tag{24}$$

In our context, the nominal model is a stochastic capital budgeting problem in which we maximize expected NPV, and we could have $f(s, \xi^{\sigma}) = NPV(s, \xi^{\sigma})$. In this context "$s \in S$" simply indicates the constraints that a prioritized solution must satisfy, wherein we prioritize project selection subject to uncertainty in costs and the NPV of each project as well as uncertainty in resource availability. The goal is to prioritize so as to maximize expected NPV, assuming that nominal distribution, specified by the probability mass function $q^{\sigma}, \sigma \in \Sigma$.

We suppose that ξ is a discrete random variable with finite sample space Ω, so that $\xi^{\omega}, \omega \in \Omega$ enumerates all possible realizations. We further suppose that we only have observations of $\xi^{\sigma}, \sigma \in \Sigma \subset \Omega$, i.e., possibly a strict subset, which may arise in a data-driven setting. In such a data-driven setting we could have, for example, probability mass $q^{\sigma} = 1/|\Sigma|$ for all $\sigma \in \Sigma$. A DRO variant of this nominal stochastic optimization model is then given by:

$$\max_{s \in S} \min_{p \in P} \sum_{\omega \in \Omega} p^{\omega} f(s, \xi^{\omega}) \tag{25}$$

Here, we may view this DRO model as playing a "game" against nature. First, we select $s \in S$, and then knowing s, nature selects a worst-case probability distribution, $p \in P$, to minimize the expected NPV, which we seek to maximize. We will make precise what we mean by the Distributional Uncertainty Set (DUS), denoted by P, below, but for the moment it is enough to think of the set as representing a neighborhood of probability distributions centered on the given probability mass function, q, with the radius of the neighborhood specified by parameter ε. If $\varepsilon = 0$ then the DRO model reduces to the nominal model. If ε is very large then nature will select the single worst-case scenario, e.g., the scenario with lowest budgets, highest costs, and lowest NPVs. This is too conservative to be useful (i.e. if we are living in this world, it is very likely that the plant would be uneconomical no matter what decisions are made). However, with moderate values of ε we obtain solutions that hedge against deviations from q without being excessively conservative.

Importantly, we do not view nature as malevolent, despite occasional evidence to the contrary. Rather, we use "$min_{p \in P}$" to combat over-adapting our solution to a specific assumption about the probability distribution. In this sense, DRO plays the role of a "regularizer" to combat overfitting that is analogous to regularizers used in high-dimensional statistics and statistical machine learning. Our approach to DRO requires further mathematical and intuitive development before

we can analyze solutions to the DRO problem.

5.1 Defining a Distributional Uncertainty Set via the Wasserstein Distance

By the constraints denoted by $p \in P$ we require that nature select a probability mass function, p, that is "close" to the nominal or empirical data-driven distribution q. We define the DUS as follows:

$$P = \{p : D(p, q) \leq \varepsilon, \sum_{\omega \in \Omega} p^{\omega} = 1, p^{\omega} \geq 0, \omega \in \Omega\} \tag{26}$$

where $D(p, q)$ is the distance between nature's choice, p, and the nominal data-driven distribution, q. As indicated above, the radius parameter ε governs the latitude we give nature, which in turn governs the degree of conservatism that we face when selecting decision $s \in S$.

There are multiple ways to measure the "distance", $D(p, q)$, between two probability distributions, which include the Kolmogorov-Smirnov distance, Kullback-Leibler divergence, chi-squared distances, total variation, and more general ψ-divergences. The Wasserstein distance, which is based on the idea of optimal transport, is a particularly useful way to measure such a distance in the context of distributionally robust optimization. For a distribution with known finite support, the Wasserstein distance, $D(p, q)$, between a given distribution, $q^{\sigma}, \sigma \in \Sigma$, and a given candidate robust distribution, $p^{\omega}, \omega \in \Omega$, is provided by the optimal value of the transportation problem:

$$D(q, p) = \min_{z} \sum_{\sigma \in \Sigma, \omega \in \Omega} d_{\sigma,\omega} z_{\sigma,\omega} \tag{27a}$$

$$s.t. \sum_{\omega \in \Omega} z_{\sigma,\omega} = q^{\sigma}, \sigma \in \Sigma \tag{27b}$$

$$\sum_{\sigma \in \Sigma} z_{\sigma,\omega} = p^{\omega}, \omega \in \Omega \tag{27c}$$

$$z_{\sigma,\omega} \geq 0, \sigma \in \Sigma, \omega \in \Omega \tag{27d}$$

The intuition behind this measure concerns the magnitude of probability mass, q^{σ}, that must be transported distance $d_{\sigma,\omega}$ from vector ξ^{σ} to vector ξ^{ω} via variable $z_{\sigma,\omega}$. In one extreme case, if the two sample spaces and probability mass functions coincide, i.e., $\Omega = \Sigma$ and $p^{\omega} = q^{\omega}$, and $d_{\omega,\omega} = 0$ for all $\omega \in \Omega$ then $D(p, q) = 0$.

To fully specify $D(p,q)$ we must define $d_{\sigma,\omega} = dist(\xi^\sigma, \xi^\omega)$. To do so we can select $dist(\cdot,\cdot)$, for example, to be the two-norm distance, or a more general η-norm distance, between the vectors, ξ^σ and ξ^ω, i.e., $dist(\cdot,\cdot) = ||\xi^\sigma - \xi^\omega||_\eta$.

With the Wasserstein distance, if we are given distribution, q, we can then define:

$$P = \{p : D(p,q) \leq \varepsilon, \sum_{\omega\in\Omega} p^\omega = 1, p^\omega \geq 0, \omega \in \Omega\} \tag{28}$$

for a given radius ε. Here, we think of P as a ball, or neighborhood, of probability distributions centered on q, where the neighborhood has radius ε. With $\Sigma \subset \Omega$, if $\varepsilon = 0$ then P is the singleton q, and larger values of ε lead to increasingly large neighborhoods. In the context of robust optimization, if $\varepsilon = 0$ then we will simply be solving the nominal stochastic optimization model, and as ε grows large we will consider increasingly conservative models.

We can now represent the set P via the following so-called extended-variable set of constraints:

$$\sum_{\sigma\in\Sigma,\omega\in\Omega} d_{\sigma,\omega} z_{\sigma,\omega} \leq \varepsilon \tag{29a}$$

$$\sum_{\omega\in\Omega} z_{\sigma,\omega} = q^\sigma, \sigma \in \Sigma \tag{29b}$$

$$\sum_{\sigma\in\Sigma} z_{\sigma,\omega} = p^\omega, \omega \in \Omega \tag{29c}$$

$$z_{\sigma,\omega} \geq 0, \sigma \in \Sigma, \omega \in \Omega \tag{29d}$$

5.2 Towards a Computationally Tractable Reformulation

Due to the max min construct in the DRO model, the model is not amenable to direct solution via optimization software. So, we reformulate the model to facilitate computation. For the moment let $s \in S$ be fixed so that $f(s,\xi^\omega)$ is just a known numerical value for each $\omega \in \Omega$. Then, nature's

problem may be written:

$$\min_{p,z} \sum_{\omega \in \Omega} p^{\omega} f(s, \xi^{\omega}) \tag{30a}$$

$$s.t. \sum_{\sigma \in \Sigma, \omega \in \Omega} d_{\sigma,\omega} z_{\sigma,\omega} \leq \varepsilon : [-\gamma] \tag{30b}$$

$$\sum_{\omega \in \Omega} z_{\sigma,\omega} = q^{\sigma}, \sigma \in \Sigma : [\nu^{\sigma}] \tag{30c}$$

$$\sum_{\sigma \in \Sigma} z_{\sigma,\omega} = p^{\omega}, \omega \in \Omega : [\beta^{\omega}] \tag{30d}$$

$$z_{\sigma,\omega} \geq 0, \sigma \in \Sigma, \omega \in \Omega \tag{30e}$$

Here, γ, ν^{σ}, and β^{ω} denote dual variables. In this model, nature optimizes over z and over p to select a worst-case distribution within radius ε of q.

Taking the dual of the linear DRO program, and substituting out the dual variable $\beta^{\omega} = -f(s, \xi^{\omega})$, we obtain the following:

$$\max_{\gamma,\nu} -\gamma\varepsilon + \sum_{\sigma \in \Sigma} \nu^{\sigma} q^{\sigma} \tag{31a}$$

$$s.t. -\gamma d_{\sigma,\omega} + \nu^{\sigma} \leq f(s, \xi^{\omega}), \sigma \in \Sigma, \omega \in \Omega \tag{31b}$$

$$\gamma \geq 0 \tag{31c}$$

For a better understanding of the model, consider two extreme cases, $\varepsilon = 0$ and $\varepsilon = \infty$. If $\varepsilon = 0$ then there is no penalty in the objective function for allowing γ to grow large. As $gamma$ grows large, the constraint becomes vacuous for all $\sigma \neq \omega$; however, for $\sigma = \omega$ we have $d_{\sigma,\sigma} = ||\xi^{\sigma} - \xi^{\omega}|| = 0$, and hence the constraint reduces to $\nu^{\sigma} \leq f(s, \xi^{\omega})$, and coupled with the objective function the optimal value reduces to $\sum_{\sigma} q^{\sigma} f(s, \xi^{\sigma}) = \sum_{\omega} p^{\omega} f(s, \xi^{\omega})$, i.e., it reduces to the objective function value of the nominal stochastic optimization model, as it must with $\varepsilon = 0$.

In the other extreme, as ε grows sufficiently large we must have $\gamma = 0$ to avoid a huge penalty in the objective function. Thus, for each $\sigma \in \Sigma$, the constraint reduces to $\nu^{\sigma} \leq f(s, \xi^{\omega})$, i.e., the objective function reduces to:

$$\sum_{\sigma} \min_{\omega} f(s, \xi^{\omega}) q^{\sigma} = \min_{\omega} f(s, \xi^{\omega}) \sum_{\sigma} q^{\sigma} = \min_{\omega} f(s, \xi^{\omega}) \tag{32}$$

Again, this matches what it must: if $\varepsilon = \infty$ then nature has enough latitude to place a probability of one on the single worst-case scenario.

Specializing $s \in S$ to be the constraints for prioritization, and specializing $f(s, \xi^{\omega})$ to define the NPV under scenario $\omega \in \Omega$, the DRO variant of the stochastic capital budgeting problem is as follows:

$$\max_{x,y,\gamma,\nu} -\gamma\varepsilon + \sum_{\sigma\in\Sigma} \nu^{\sigma} q^{\sigma} \tag{33a}$$

$$s.t. -\gamma d_{\sigma,\omega} + \nu^{\sigma} \leq \sum_{i\in I} \sum_{j\in J_i} a_{ij}^{\omega} x_{ij}^{\omega}, \sigma \in \Sigma, \omega \in \Omega \tag{33b}$$

$$y_{ii'} + y_{i'i} \geq 1,\ i < i', \mathrm{i}, i' \in I \tag{33c}$$

$$\sum_{j=1}^{J_i} x_{ij}^{\omega} \geq \sum_{j=1}^{J_i} x_{i'j}^{\omega} + y_{ii'} - 1,\ i \neq i', \mathrm{i}, i' \in I, \omega \in \Omega \tag{33d}$$

$$\sum_{i\in I} \sum_{j\in J_i} \mathrm{c}_{ijkt}^{\omega} x_{ij}^{\omega} \leq b_{kt}^{\omega},\ k \in K, t \in T, \omega \in \Omega \tag{33e}$$

$$\sum_{j\in J_i} x_{ij}^{\omega} \leq 1,\ i \in I, \omega \in \Omega \tag{33f}$$

$$\gamma \geq 0 \tag{33g}$$

5.3 LOGOS Settings for DRO Problems

DRO approach is an extension for stochastic optimization approach discussed in Section 4. Both of them share the same input structures except the **<Settings>** block. In both cases, the user need to specify a collection of scenarios via **<Uncertainties>** block. The **<problem_type>** within **<Settings>** block is used to select the type of DRO problems. The currently available DRO problem types are: **'droskp'**, **'dromkp'**, and **'dromckp'**. The user can use **<radius_ambiguity>** to control the Wasserstein distance for DRO problems.

Example LOGOS input XML for DRO:

```
<Settings>
<Logos>
  <solver>cbc</solver>
  <solverOptions>
    <StochSolver>EF</StochSolver>
    <!-- epsilon radius -->
```

```
    <radius_ambiguity>0.1</radius_ambiguity>
  </solverOptions>
  <sense>maximize</sense>
  <problem_type>droskp</problem_type>
</Settings>
</Logos>
```

5.4 DRO for Single Knapsack Problem

Model Formulation:

$$\max_{x,y} \max_{\gamma,\nu} (-\gamma\varepsilon + \sum_{\sigma\in\Sigma} \nu^\sigma q^\sigma) \tag{34a}$$

$$-\gamma d_{\sigma,\omega} + \nu^\sigma \leq \sum_{i\in I} a_i^\omega x_i^\omega, \sigma \in \Sigma, \omega \in \Omega \tag{34b}$$

$$\sum_{i\in I} c_i^\omega x_i^\omega \leq b^\omega \tag{34c}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{34d}$$

$$x_i^\omega \geq x_{i'}^\omega + y_{ii'} - 1, i \neq i' \tag{34e}$$

$$x_i^\omega, y_{ii'}^\omega \in 0, 1 \tag{34f}$$

$$\gamma \geq 0 \tag{34g}$$

See next section 5.5 for the example of LOGOS input file, since multi-dimensional Knapsack problem is just a simple extension of a single-dimensional Knapsack problem, and both of them belong to the same **<problem_type>**: **'droskp'**.

5.5 DRO for Multi-Dimensional Knapsack Problem

Model Formulation:

$$\max_{x,y} \max_{\gamma,\nu} (-\gamma\varepsilon + \sum_{\sigma\in\Sigma} \nu^\sigma q^\sigma) \tag{35a}$$

$$-\gamma d_{\sigma,\omega} + \nu^\sigma \leq \sum_{i\in I} a_i^\omega x_i^\omega, \sigma \in \Sigma, \omega \in \Omega \tag{35b}$$

$$\sum_{i\in I} c_{it}^\omega x_i^\omega \leq b_t^\omega, t \in T \tag{35c}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{35d}$$

$$x_i^\omega \geq x_{i'}^\omega + y_{ii'} - 1, i \neq i' \tag{35e}$$

$$x_i^\omega, y_{ii'}^\omega \in 0, 1 \tag{35f}$$

$$\gamma \geq 0 \tag{35g}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
    </investments>
    <time_periods>
      1,2,3,4,5
    </time_periods>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      2.315,0.824,22.459,60.589,0.667,5.173,4.003,0.582,0.122,
      -2.870,-0.102,-0.278,-0.322,-3.996,-0.246,-20.155
    </net_present_values>
    <costs index="investments,␣time_periods">
      0.219,0.257,0.085,0.0,0.0,
      0.0,0.0,0.122,0.103,0.013,
      5.044,1.839,0.0,0.0,0.0,
      6.74,6.134,10.442,0.0,0.0,
      0.425,0.0,0.0,0.0,0.0,
      2.125,2.122,0.0,0.0,0.0,
      2.387,0.19,0.012,2.383,0.192,
```

```
    0.0,0.95,0.0,0.0,0.0,
    0.03,0.03,0.688,0.0,0.0,
    0,0.2,0.763,0.739,2.539,
    0.081,0.032,0,0,0,
    0.3,0,0,0,0,
    0.347,0,0,0,0,
    4.025,0.297,0,0,0,
    0.095,0.095,0.095,0,0,
    5.487,5.664,0.5,6.803,6.778
  </costs>
  <available_capitals index="time_periods">
    18,18,18,18,18
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
    <totalScenarios>10</totalScenarios>
    <probabilities>
      0.012, 0.019, 0.032, 0.052, 0.086, 0.142, 0.235, 0.188,
        0.141, 0.093
    </probabilities>
    <!--
      scenarios is ordered by numberScenarios *
        parametersIndex,
      the number of scenarios is determined by
      the number of elements in <probabilities>, for this case:
      numberScenarios * time_periods = 10 * 5
    -->
    <scenarios>
      11, 11, 11, 11, 11,
      12, 12, 12, 12, 12,
      13, 13, 13, 13, 13,
      14, 14, 14, 14, 14,
      15, 15, 15, 15, 15,
      16, 16, 16, 16, 16,
      17, 17, 17, 17, 17,
      18, 18, 18, 18, 18,
      19, 19, 19, 19, 19,
      20, 20, 20, 20, 20
    </scenarios>
  </available_capitals>
```

```
  </Uncertainties>

  <Settings>
    <mandatory>10,11,12,13,14,15,16</mandatory>
    <solver>cbc</solver>
    <solverOptions>
      <StochSolver>EF</StochSolver>
      <!-- epsilon radius -->
      <radius_ambiguity>0.1</radius_ambiguity>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>droskp</problem_type>
  </Settings>
</Logos>
```

5.6 DRO for Multiple Knapsack Problem

Model Formulation:

$$\max_{x,y} \max_{\gamma,\nu} (-\gamma\varepsilon + \sum_{\sigma\in\Sigma} \nu^{\sigma} q^{\sigma}) \tag{36a}$$

$$-\gamma d_{\sigma,\omega} + \nu^{\sigma} \leq \sum_{i\in I} \sum_{m\in M} a_i^{\omega} x_{im}^{\omega}, \sigma \in \Sigma, \omega \in \Omega \tag{36b}$$

$$\sum_{i\in I} c_i^{\omega} x_{im}^{\omega} \leq b_m^{\omega},\ m \in M, \omega \in \Omega \tag{36c}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{36d}$$

$$\sum_{m=1}^{M} x_{im}^{\omega} \geq \sum_{m=1}^{M} x_{i'm}^{\omega} + y_{ii'} - 1,\ i \neq i', \mathrm{i}, i' \in I, \omega \in \Omega \tag{36e}$$

$$x_{i,m}^{\omega}, y_{ii'}^{\omega} \in 0, 1 \tag{36f}$$

$$\gamma \geq 0 \tag{36g}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10
```

```
  </investments>
  <capitals>
    unit_1, unit_2
  </capitals>
</Sets>

<Parameters>
  <net_present_values index="investments">
    78, 35, 89, 36, 94, 75, 74, 79, 80, 16
  </net_present_values>
  <costs index="investments">
    18, 9, 23, 20, 59, 61, 70, 75, 76, 30
  </costs>
  <available_capitals index="capitals">
    103, 156
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
    <totalScenarios>10</totalScenarios>
    <probabilities>
      0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
    </probabilities>
    <scenarios>
      101, 154,
      102, 155,
      103, 156,
      104, 157,
      105, 158,
      106, 159,
      107, 160,
      108, 161,
      109, 162,
      110, 163
    </scenarios>
  </available_capitals>
</Uncertainties>

<Settings>
  <solver>cbc</solver>
  <solverOptions>
```

```
      <StochSolver>EF</StochSolver>
      <!-- epsilon radius -->
      <radius_ambiguity>0.1</radius_ambiguity>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>dromkp</problem_type>
  </Settings>
</Logos>
```

5.7 DRO for Multiple-Choice Knapsack Problem

Model Formulation:

$$\max_{x,y} \max_{\gamma,\nu} (-\gamma\varepsilon + \sum_{\sigma\in\Sigma} \nu^\sigma q^\sigma) \tag{37a}$$

$$-\gamma d_{\sigma,\omega} + \nu^\sigma \leq \sum_{i\in I}\sum_{j\in J_i} a^\omega_{ij} x^\omega_{ij}, \sigma \in \Sigma, \omega \in \Omega \tag{37b}$$

$$\sum_{i\in I}\sum_{j\in J_i} \mathrm{c}^\omega_{ijkt} x^\omega_{ij} \leq b^\omega_{kt},\ k \in K, t \in T, \omega \in \Omega \tag{37c}$$

$$y_{ii'} + y_{i'i} \geq 1, i < i' \tag{37d}$$

$$\sum_{j=1}^{J_i} x^\omega_{ij} \geq \sum_{j=1}^{J_i} x^\omega_{i'j} + y_{ii'} - 1,\ i \neq i', \mathrm{i}, i' \in I, \omega \in \Omega \tag{37e}$$

$$x^\omega_{i,j}, y^\omega_{ii'} \in 0, 1 \tag{37f}$$

$$\gamma \geq 0 \tag{37g}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
    </investments>
    <options index='investments'>
      1;
      1;
      1;
      1,2,3;
```

```
    1,2,3,4;
    1,2,3,4,5,6,7;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1;
    1
  </options>
</Sets>

<Parameters>
  <net_present_values index='options'>
    2.046
    2.679
    2.489
    2.61
    2.313
    1.02
    3.013
    2.55
    3.351
    3.423
    3.781
    2.525
    2.169
    2.267
    2.747
    4.309
    6.452
    2.849
    7.945
    2.538
    1.761
    3.002
    3.449
    2.865
```

```
    3.999
    2.283
    0.9
    8.608
  </net_present_values>
  <costs index='options'>
    36538462
    83849038
    4615385
    2788461538
    2692307692
    5480769231
    1634615385
    2981730768
    7211538462
    9038461538
    649038462
    650000000
    216346154
    212500000
    3076923077
    3942307692
    1144230769
    675721154
    1442307692
    99711538
    4807692
    123076923
    138461538
    86538462
    108653846
    75092404
    6413462
    147932692
  </costs>
  <available_capitals>
    15E9
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
```

```
      <totalScenarios>3</totalScenarios>
      <probabilities>
        0.2,0.6,0.2
      </probabilities>
      <scenarios>
        5E9,10E9,15E9
      </scenarios>
    </available_capitals>
  </Uncertainties>

  <Settings>
    <solver>glpk</solver>
    <solverOptions>
      <StochSolver>EF</StochSolver>
      <!-- epsilon radius -->
      <radius_ambiguity>0.1</radius_ambiguity>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>dromckp</problem_type>
  </Settings>
</Logos>
```

6 Risk-Based Stochastic Capital Budgeting using Conditional Value-at-Risk

Value-at-Risk (VaR) is currently used by finance businesses to indicate the percentiles of loss distributions. For instance, 95%-VaR is an upper estimate of losses which is exceeded with 5% probability. The popularity of VaR is mostly related to a simple and easy to understand representation of high losses. However, VaR may have undesirable mathematical characteristics such as a lack of subadditivity and convexity [4, 5] . Another alternative percentile risk measure is called Conditional Value-at-Risk (CVaR), which has more attractive properties than VaR, such as subadditive and convex. CVaR also called mean excess loss, mean shortfall, or tail VaR, is defined as the expected loss exceeding VaR. In general, CVaR is the weighted average of VaR and losses exceeding VaR. In this section, we will focus on using CVaR for capital budgeting problem.

6.1 Definitions of VaR and CVaR

Let X be a random variable with a cumulative distribution function $F(z) = PX \leq z$. It can be useful to think of X as a "loss" or more generally a variable such that large values need to be avoid. The VaR of X with confidence level α (e.g., $\alpha = 0.9$ is:

$$VaR_\alpha(X) = \min z | F_X(z) \geq \alpha \tag{38}$$

which is equivalent to $VaR_\alpha(X) = F_X^{-1}(\alpha)$ if X is a continuous random variable. By this definition, $VaR_\alpha(X)$ is a (lower) α-percentile of the random variable X. An alternative measure of risk is CVaR. Here, $CVaR_\alpha(X)$ is the conditional expectation of X given that $X \geq VaR_\alpha(X)$. Figure 1 shows the relationship between these two measures of risk.

The typical definition of $CVaR_\alpha(X)$ is $CVaR_\alpha(X) = EX|X > VaR_\alpha(X)$. There are alternative ways to define this measure, which are mathematically equivalent. Rockafellar and Uryasev in [6] (see also [7]) defines CVaR as:

$$CVaR_\alpha(X) = \min_u u + 1/(1-\alpha)E[X-u]^+ \tag{39}$$

where $[X - u] = \max(X - u, 0)$. Here, variable u is simply an auxiliary decision variable whose optimal value turns out to be $CVaR_\alpha(X)$. The above definition is particularly useful for computation in the context of optimization.

Researchers have argued for using CVaR over VaR as a measure of risk. Theoretically, CVaR satisfies the assumptions of a so-called coherent risk measure, and VaR does not. In simpler terms,

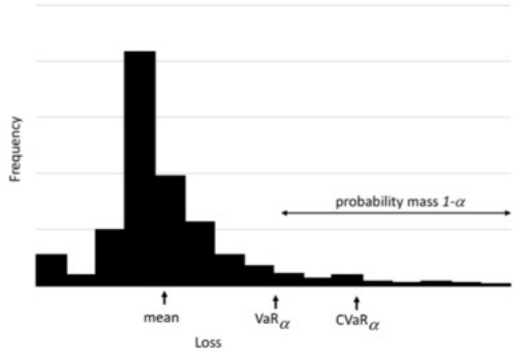


Figure 1. Relationship between value-at-risk and conditional value-at-risk.

minimizing VaR is concerned with the numerical value of the 95-th percentile (say) of the loss, but it does not care about the magnitude of larger losses. CVaR takes these magnitudes into account.

6.2 CVaR in Capital Budgeting

In this section, an explicit risk measure is constructed using a weighted combination of expectation and CVaR. This approach allows us to parametrically vary the weight on maximizing expected NPV versus penalizing solutions that yield low-NPV scenarios, and we denote the weight by λ with $0 \leq \lambda \leq 1$. Let $NPV(s, \xi)$ denote the net present value under a prioritization decision specified by decision s, and under a realization of the budget and profit of each project, denoted by ξ. Then we seek to solve the following optimization model:

$$\max_{s \in S} (1 - \lambda) E[NPV(s, \xi)] - \lambda CVaR_{\alpha}[-NPV(s, \xi)] \tag{40}$$

When $\lambda = 0$ the model reduces to stochastic optimization model as discussed in Section 4; i.e., we seek a prioritization decision, s, to maximize expected NPV, where "$s \in S$" simply indicates the constraints that a prioritized solution must satisfy. $CVaR_{\alpha}[X]$ is typically applied to a random variable, X, which represents a loss; i.e., we seek to avoid large values of X. In this context, let $VaR_{\alpha}[X]$ denote the α-level quantile of X. Thus, if $\alpha = 0.75$ then $VaR_0.75[X]$ is the value such that 75% of the realizations of X have lower values of loss. Suppose for simplicity that $NPV(s, \xi)$ values are positive. Large values of $NPV(s, \xi)$ are good, and hence large values of $-NPV(s, \xi)$ (i.e., those closer to zero) are bad. Using the definition of $CVaR_{\alpha}[X] = E[X|X > VaR_{\alpha}[X]]$ we thus have that the conditional value-at-risk is the expected value of loss, given that the loss exceeds a certain percentile. So, when $\lambda = 1$ we seek to minimize the expected value of NPV given that

they fall below a threshold. More generally, values of λ between 0 and 1 seek a trade-off between reward and risk, captured by expected NPV and CVaR, respectively.

The full mathematical optimization model of CVaR for capital budgeting problem is as follows:

$$\max_{x,y,\nu,u}(1-\lambda)\sum_{\omega\in\Omega}q^{\omega}\sum_{i\in I}\sum_{j\in J_i}a_{ij}^{\omega}x_{ij}^{\omega}-\lambda[u+1/(1-\alpha)\sum_{\omega\in\Omega}q^{\omega}\nu^{\omega}] \tag{41a}$$

$$\nu^{\omega}\geq-\sum_{i\in I}\sum_{j\in J_i}a_{ij}^{\omega}x_{ij}^{\omega}-u,\omega\in\Omega \tag{41b}$$

$$y_{ii'}+y_{i'i}\geq 1,\ i<i^{'},\mathrm{i},i^{'}\in I \tag{41c}$$

$$\sum_{j=1}^{J_i}x_{ij}^{\omega}\geq\sum_{j=1}^{J_i}x_{i'j}^{\omega}+y_{ii'}-1,\ i\neq i^{'},\mathrm{i},i^{'}\in I,\omega\in\Omega \tag{41d}$$

$$\sum_{i\in I}\sum_{j\in J_i}\mathrm{c}_{ijkt}^{\omega}x_{ij}^{\omega}\ \leq b_{kt}^{\omega},\ k\in K,t\in T,\omega\in\Omega \tag{41e}$$

$$\sum_{j\in J_i}x_{ij}^{\omega}\leq 1,\ i\in I,\omega\in\Omega \tag{41f}$$

$$y_{ii'},x_{ij}^{\omega}\in 0,1 \tag{41g}$$

$$\nu^{\omega}\geq 0,\omega\in\Omega \tag{41h}$$

6.3 LOGOS Settings for CVaR Problems

CVaR approach is an extension for stochastic optimization approach discussed in Section 4. Both of them share the same input structures except the **<Settings>** block. In both cases, the user need to specify a collection of scenarios via **<Uncertainties>** block. The **<problem_type>** within **<Settings>** block is used to select the type of CVaR problems. The currently available CVaR problem types are: **'cvarskp'**, **'cvarmkp'**, and **'cvarmckp'**. The user can use **<risk_aversion>** (i.e. λ) and **<confidence_level>** (i.e., α) to control CVaR problem.

Example LOGOS input XML for CVaR:

```
<Settings>
<Logos>
  <solver>cbc</solver>
  <solverOptions>
    <StochSolver>EF</StochSolver>
    <risk_aversion>0.1</risk_aversion>
    <confidence_level>0.95</confidence_level>
```

```
  </solverOptions>
  <sense>maximize</sense>
  <problem_type>cvarskp</problem_type>
</Settings>
</Logos>
```

6.4 CVaR for Single Knapsack Problem

Model Formulation:

$$\max_{x,y,\nu,u}(1-\lambda)\sum_{\omega\in\Omega}q^{\omega}\sum_{i\in I}a_i^{\omega}x_i^{\omega}-\lambda[u+1/(1-\alpha)\sum_{\omega\in\Omega}q^{\omega}\nu^{\omega}] \tag{42a}$$

$$\nu^{\omega}\geq-\sum_{i\in I}a_i^{\omega}x_i^{\omega}-u,\omega\in\Omega \tag{42b}$$

$$\sum_{i\in I}c_i^{\omega}x_i^{\omega}\leq b^{\omega},\omega\in\Omega \tag{42c}$$

$$y_{ii'}+y_{i'i}\geq 1, i<i' \tag{42d}$$

$$x_i^{\omega}\geq x_{i'}^{\omega}+y_{ii'}-1, i\neq i' \tag{42e}$$

$$x_i^{\omega}, y_{ii'}^{\omega}\in 0,1 \tag{42f}$$

$$\nu^{\omega}\geq 0,\omega\in\Omega,\lambda\in[0,1] \tag{42g}$$

See next section 6.5 for the example of LOGOS input file, since multi-dimensional Knapsack problem is just a simple extension of a single-dimensional Knapsack problem, and both of them belong to the same **<problem_type>**: **'cvarskp'**.

6.5 CVaR for Multi-Dimensional Knapsack Problem

Model Formulation:

$$\max_{x,y,\nu,u}(1-\lambda)\sum_{\omega\in\Omega}q^{\omega}\sum_{i\in I}a_i^{\omega}x_i^{\omega}-\lambda[u+1/(1-\alpha)\sum_{\omega\in\Omega}q^{\omega}\nu^{\omega}] \tag{43a}$$

$$\nu^{\omega}\geq-\sum_{i\in I}a_i^{\omega}x_i^{\omega}-u,\omega\in\Omega \tag{43b}$$

$$\sum_{i\in I}c_{it}^{\omega}x_i^{\omega}\leq b_t^{\omega},t\in T \tag{43c}$$

$$y_{ii'}+y_{i'i}\geq 1,i<i' \tag{43d}$$

$$x_i^{\omega}\geq x_{i'}^{\omega}+y_{ii'}-1,i\neq i' \tag{43e}$$

$$x_i^{\omega},y_{ii'}^{\omega}\in 0,1 \tag{43f}$$

$$\nu^{\omega}\geq 0,\omega\in\Omega,\lambda\in[0,1] \tag{43g}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
    </investments>
    <time_periods>
      1,2,3,4,5
    </time_periods>
  </Sets>

  <Parameters>
    <net_present_values index="investments">
      2.315,0.824,22.459,60.589,0.667,5.173,4.003,0.582,0.122,
      -2.870,-0.102,-0.278,-0.322,-3.996,-0.246,-20.155
    </net_present_values>
    <costs index="investments,␣time_periods">
      0.219,0.257,0.085,0.0,0.0,
      0.0,0.0,0.122,0.103,0.013,
      5.044,1.839,0.0,0.0,0.0,
      6.74,6.134,10.442,0.0,0.0,
      0.425,0.0,0.0,0.0,0.0,
      2.125,2.122,0.0,0.0,0.0,
```

```
    2.387,0.19,0.012,2.383,0.192,
    0.0,0.95,0.0,0.0,0.0,
    0.03,0.03,0.688,0.0,0.0,
    0,0.2,0.763,0.739,2.539,
    0.081,0.032,0,0,0,
    0.3,0,0,0,0,
    0.347,0,0,0,0,
    4.025,0.297,0,0,0,
    0.095,0.095,0.095,0,0,
    5.487,5.664,0.5,6.803,6.778
  </costs>
  <available_capitals index="time_periods">
    18,18,18,18,18
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
    <totalScenarios>10</totalScenarios>
    <probabilities>
      0.012, 0.019, 0.032, 0.052, 0.086, 0.142, 0.235, 0.188,
        0.141, 0.093
    </probabilities>
    <scenarios>
      11, 11, 11, 11, 11,
      12, 12, 12, 12, 12,
      13, 13, 13, 13, 13,
      14, 14, 14, 14, 14,
      15, 15, 15, 15, 15,
      16, 16, 16, 16, 16,
      17, 17, 17, 17, 17,
      18, 18, 18, 18, 18,
      19, 19, 19, 19, 19,
      20, 20, 20, 20, 20
    </scenarios>
  </available_capitals>
</Uncertainties>

<Settings>
  <mandatory>10,11,12,13,14,15,16</mandatory>
  <solver>cbc</solver>
  <solverOptions>
```

```
      <StochSolver>EF</StochSolver>
      <!-- lambda for risk aversion -->
      <risk_aversion>0.1</risk_aversion>
      <!-- confidence level -->
      <confidence_level>0.95</confidence_level>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>cvarskp</problem_type>
  </Settings>
</Logos>
```

6.6 CVaR for Multiple Knapsack Problem

Model Formulation:

$$\max_{x,y,\nu,u}(1-\lambda)\sum_{\omega\in\Omega}q^{\omega}\sum_{m\in M}\sum_{i\in I}a_i^{\omega}x_{i,m}^{\omega}-\lambda[u+1/(1-\alpha)\sum_{\omega\in\Omega}q^{\omega}\nu^{\omega}] \tag{44a}$$

$$\nu^{\omega}\geq-\sum_{m\in M}\sum_{i\in I}a_i^{\omega}x_{i,m}^{\omega}-u,\omega\in\Omega \tag{44b}$$

$$\sum_{i\in I}c_i^{\omega}x_{im}^{\omega}\ \leq b_m^{\omega},\ m\in M,\omega\in\Omega \tag{44c}$$

$$y_{ii'}+y_{i'i}\geq 1, i<i' \tag{44d}$$

$$\sum_{m=1}^{M}x_{im}^{\omega}\geq\sum_{m=1}^{M}x_{i'm}^{\omega}+y_{ii'}-1,\ i\neq i^{'},\mathrm{i},i^{'}\in I,\omega\in\Omega \tag{44e}$$

$$\sum_{m=1}^{M}x_{im}^{\omega}\leq 1 \tag{44f}$$

$$x_{im}^{\omega},y_{ii'}^{\omega}\in 0,1 \tag{44g}$$

$$\nu^{\omega}\geq 0,\omega\in\Omega,\lambda\in[0,1] \tag{44h}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1,2,3,4,5,6,7,8,9,10
    </investments>
```

```
  <capitals>
    unit_1, unit_2
  </capitals>
</Sets>

<Parameters>
  <net_present_values index="investments">
    78, 35, 89, 36, 94, 75, 74, 79, 80, 16
  </net_present_values>
  <costs index="investments">
    18, 9, 23, 20, 59, 61, 70, 75, 76, 30
  </costs>
  <available_capitals index="capitals">
    103, 156
  </available_capitals>
</Parameters>

<Uncertainties>
  <available_capitals>
    <totalScenarios>10</totalScenarios>
    <probabilities>
      0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
    </probabilities>
    <scenarios>
      101, 154,
      102, 155,
      103, 156,
      104, 157,
      105, 158,
      106, 159,
      107, 160,
      108, 161,
      109, 162,
      110, 163
    </scenarios>
  </available_capitals>
</Uncertainties>

<Settings>
  <solver>cbc</solver>
  <solverOptions>
    <StochSolver>EF</StochSolver>
```

```
      <risk_aversion>1.0</risk_aversion>
      <confidence_level>0.95</confidence_level>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>cvarmkp</problem_type>
  </Settings>
</Logos>
```

6.7 CVaR for Multiple-Choice Knapsack Problem

Model Formulation:

$$\max_{x,y,\nu,u}(1-\lambda)\sum_{\omega\in\Omega}q^{\omega}\sum_{j\in J_i}\sum_{i\in I}a_{ij}^{\omega}x_{ij}^{\omega}-\lambda[u+1/(1-\alpha)\sum_{\omega\in\Omega}q^{\omega}\nu^{\omega}] \tag{45a}$$

$$\nu^{\omega}\geq-\sum_{j\in J_i}\sum_{i\in I}a_{ij}^{\omega}x_{ij}^{\omega}-u,\omega\in\Omega \tag{45b}$$

$$\sum_{i\in I}\sum_{j\in J_i}\mathrm{c}_{ijkt}^{\omega}x_{ij}^{\omega}\leq b_{kt}^{\omega},\ k\in K,t\in T,\omega\in\Omega \tag{45c}$$

$$y_{ii'}+y_{i'i}\geq 1,i<i' \tag{45d}$$

$$\sum_{j=1}^{J_i-1}x_{ij}^{\omega}\geq\sum_{j=1}^{J_i-1}x_{i'j}^{\omega}+y_{ii'}-1,\ i\neq i^{'},\mathrm{i},i^{'}\in I,\omega\in\Omega,andi\neq i' \tag{45e}$$

$$\sum_{j=1}^{J_i}x_{ij}^{\omega}=1 \tag{45f}$$

$$x_{i,j}^{\omega},y_{ii'}^{\omega}\in 0,1 \tag{45g}$$

$$\nu^{\omega}\geq 0,\omega\in\Omega,\lambda\in[0,1] \tag{45h}$$

Example LOGOS input XML:

```
<Logos>
  <Sets>
    <investments>
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
    </investments>
    <options index='investments'>
      1;
```

```
      1;
      1;
      1,2,3;
      1,2,3,4;
      1,2,3,4,5,6,7;
      1;
      1;
      1;
      1;
      1;
      1;
      1;
      1;
      1;
      1
    </options>
  </Sets>

  <Parameters>
    <net_present_values index='options'>
      2.046
      2.679
      2.489
      2.61
      2.313
      1.02
      3.013
      2.55
      3.351
      3.423
      3.781
      2.525
      2.169
      2.267
      2.747
      4.309
      6.452
      2.849
      7.945
      2.538
      1.761
```

```
      3.002
      3.449
      2.865
      3.999
      2.283
      0.9
      8.608
    </net_present_values>
    <costs index='options'>
      36538462
      83849038
      4615385
      2788461538
      2692307692
      5480769231
      1634615385
      2981730768
      7211538462
      9038461538
      649038462
      650000000
      216346154
      212500000
      3076923077
      3942307692
      1144230769
      675721154
      1442307692
      99711538
      4807692
      123076923
      138461538
      86538462
      108653846
      75092404
      6413462
      147932692
    </costs>
    <available_capitals>
      15E9
    </available_capitals>
  </Parameters>
```

```
  <Uncertainties>
    <available_capitals>
      <totalScenarios>3</totalScenarios>
      <probabilities>
        0.2,0.6,0.2
      </probabilities>
      <scenarios>
        5E9,10E9,15E9
      </scenarios>
    </available_capitals>
  </Uncertainties>

  <Settings>
    <solver>glpk</solver>
    <solverOptions>
      <StochSolver>EF</StochSolver>
      <risk_aversion>0.1</risk_aversion>
      <confidence_level>0.95</confidence_level>
    </solverOptions>
    <sense>maximize</sense>
    <problem_type>cvarmckp</problem_type>
  </Settings>
</Logos>
```

7 Plugin for the RAVEN Code

The deterministic model can be repeatedly solved by changing the input values (i.e., **<available capitals>** [b_{kt}] and **<net_present_values>** [a_{ij}]). This will allow for what-if sensitivity analysis to identify the crucial drivers behind the optimal project selection decision. Monte Carlo simulation permits a powerful variant of this approach in which we model a_{ij} and b_{kt} as random variables, sample from their distributions, and perform a form of uncertainty quantification in terms of the resulting distributions governing the binary decisions selected, x_{ij}, and the overall NPV of the selected portfolio.

Example RAVEN input **<ExternalModel>** XML:

```
<Models>
  <ExternalModel name="singleKnapsack"
     subType="LOGOS.CapitalInvestmentModel">
    <variables>available_capitals,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,
      MaxNPV</variables>
    <ModelData>
      <Sets>
        <investments>
          i1,i2,i3,i4,i5,i6,i7,i8,i9,i10
        </investments>
      </Sets>
      <Parameters>
        <net_present_values index="investments">
          18,20,17,19,25,21,27,23,25,24
        </net_present_values>
        <costs index="investments">
          1,3,7,4,8,9,6,10,2,5
        </costs>
        <available_capitals>
          15
        </available_capitals>
      </Parameters>
      <Settings>
        <solver>glpk</solver>
        <sense>maximize</sense>
      </Settings>
    </ModelData>
  </ExternalModel>
</Models>
```

As the name suggests, an external model is an entity that is embedded in the RAVEN code at run time. This object allows the user to import the LOGOS module that will be treated as a predefined internal RAVEN object. In other words, the **External Model** will be treated by RAVEN as a normal external model. Check the RAVEN user manual for a more detailed description. **Note:** The value for attribute `subType` should always be `'LOGOS.CapitalInvestmentModel'`.

`<variables>` specifies a list of variable names that need to match variables used/defined in the LOGOS model. In the above example, the variable `'available_capitals'` is sampled by RAVEN, and its value is used by LOGOS instead of using the default values specified by `<ModelData>`. The decision variables `'i1,i2,i3,i4,i5,i6,i7,i8,i9,i10'` and the objective variable `'MaxNPV'` are collected from the output of the LOGOS model, and can be stored in the RAVEN data objects. The XML node `<ModelData>` is used to specify the LOGOS optimization problem, which should be consistent with the LOGOS input XML file.

7.1 Test Automation

Automated regression testing is a development methodology generally used to verify the correctness and performance of software after each modification. This methodology is integrated directly into GitHub and GitLab for RAVEN and RAVEN-supported plugins. In this case, testing is performed automatically as part of the continuous integration system (CIS) process whenever a user commits a change to the repository. Tests of changes across multiple platforms are executed with each pull request. Results from each test execution are maintained in an approved records repository in the CIS database, along with results from the timing executions.

7.2 Testing System Prerequisites

The module test system consists of scripts written in Bash shell language and Python. It may be used on any platform supported by RAVEN (i.e. Linux, Mac, and Windows). The following conditions must be satisfied for the module test system to function properly:

- RAVEN should be installed and updated. This is because RAVEN is used to run the module tests;
- The system running the tests must be configured with the software prerequisites necessary to build and run RAVEN. These include a Python interpreter, Python libraries (h5py, matplotlib, numpy, scipy, and scikit-learn), and development tools (C++ compiler, Miniconda package manager for Python, and Git source code control);
- RAVEN must be built with the appropriate compiler before it can be used to run the tests;

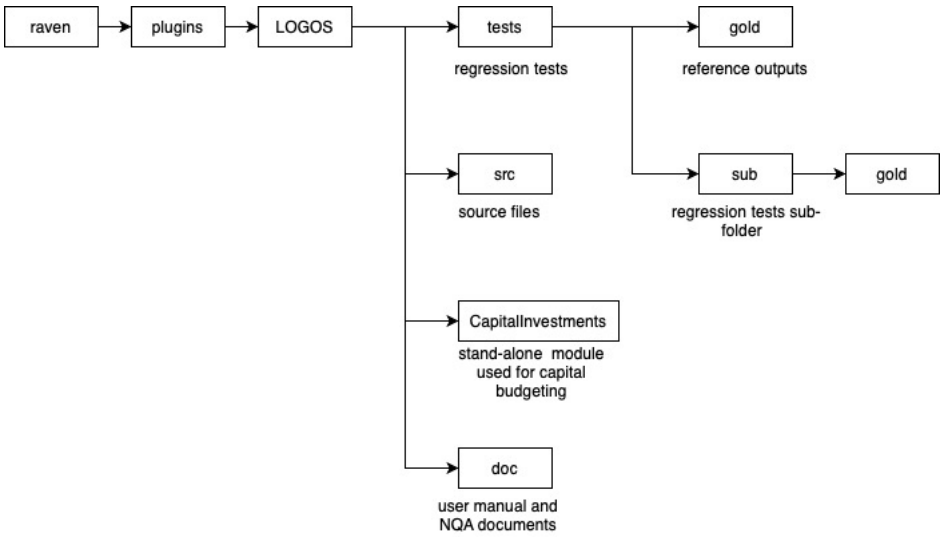


Figure 2. Folder tree of the Logos repository.

- The LOGOS submodule must be initialized and fully updated. Additional prerequisites include PYOMO, glpk, and coincbc;

7.3 Test Location and Definition

LOGOS is a RAVEN-supported plugins and managed by Git. RAVEN plugins afford an option to associate a workflow or a set of RAVEN external models to RAVEN without having them included in the RAVEN main repository. The benefits include modularity, access restriction, and regression testing for compatibility with RAVEN as it continues to grow. In this case, we are able to treat this repository as separate, yet still be able to use one from within the other. The main structure of the LOGOS repository is shown in Figure 2. The folder *tests* contains the input of the tests, in addition to the corresponding output (in the gold folder) used for assurance that the behavior of the code is not changed for new modifications. These tests can also be collected in subfolders based on their characteristics.

The RAVEN repository contains a complete testing system for providing regression testing for itself and its plugins. The LOGOS module tests are defined in the same manner as for RAVEN. A single test consists of a RAVEN input file along with associated data needed to perform that run. This can include input data, external models, and Python files. These may be placed in the *tests* directory or a related subdirectories. Each directory that contains tests to be run by the framework must contain a test specification file named “tests”. The syntax of these files is defined by the RAVEN test framework, which controls how each test is run and sets the criteria used to determine whether it passed or not. An example of a test specification file is presented here:

```
[Tests]
  [./skp_optimization]
    type = 'RavenFramework'
    input = 'test_skp.xml'
    UnorderedCsv = 'skp/test_skp.csv'
  [../]
[]
```

In the above example, one test, named "skp_optimization", is defined using the RAVEN test module (defined in the test file as "type = 'RavenFramework'"). Comparison criteria are also defined in the "tests" file. In most cases, one or more output files generated by running the specified input file with RAVEN are compared against a gold standard provided by the developer and stored in the repository. Typically, comparisons are performed on numeric values contained in CSV files up to a defined tolerance. When these file comparisons are specified by the test developer, reference files must have the same name and be placed in the gold subdirectory below that containing the "tests" file.

7.4 Running the Tests

The integrated tests can be run separately, as indicated by following this pathway:

```
path/to/raven/raven_framework path/to/LOGOS/tests/TestName
```

7.5 Continuous Integration System (CIVET)

CIVET, developed at INL, is used for continuous integration, verification, enhancement, and testing of RAVEN and LOGOS. Each time a developer proposes modification of the contents of the LOGOS repository, CIVET will cause the automated tests to be run on the modified version. These tests must all pass before a proposed change can become part of the official repository. In this way, the LOGOS project is protected from the accidental introduction of flaws into software that required such a significant investment of resources to develop.

8 SSC Cashflow and NPV Models

Imagine a portfolio of candidate projects, in which some of the decisions involve either replacing an item now or postponing replacement and facing potentially higher maintenance and replacement costs. We choose to assume that the item must either be replaced now or in the future, and it is in this context we describe the appropriate cashflow calculations. We then extend the discussion to allow the planned replacement to occur in year 2 or 3, for example, rather than in year 1.

We assume that doing nothing is not an option, since the items under consideration are of significant importance and, if not replaced in due course, would impose an unacceptable risk to either safety or production.

Notation:

- p : probability of item failure for one year
- C_P : cost of planned replacement
- C_U : cost of unplanned replacement
- C_D : cost of shutdown per day
- D : number of days plant is off-line, if a shutdown occurs
- N : number of years
- R : discount rate

We could incorporate additional parameters, such as weekly or monthly inspection costs, fixed costs of shutdown in addition to the daily costs specified above, etc. That said, the setting we describe allows us to illustrate key ideas in the cashflow calculations for computing NPV.

We further assume that, if we do not replace the item, its failure time is a random variable that follows a geometric distribution where the probability of failure in one year is denoted by p (i.e., the probability of survival over one year is $1 - p$). Thus, if the plant faces a 20-year decision period, the probability of survival up to year t is $(1 - p)^t$, and the probability of failing in year t is given by $p\,(1 - p)^{t-1}$.

Here is pedantic but useful construct for thinking about the calculations that follow is to visualize a *coin flip* for each year, yielding a *fail* or *no fail* event for that given year. Immediately after the coin flip, appropriate costs are incurred. In other words, this discrete view of time with Bernoulli trials is useful to simplify the logic of the calculations, rather than viewing time as a continuum and teasing out *what happened when* during a given year.

If the item is not replaced today (time $t = 1$), we can compute the expected replacement cost in any year $t = 1, 2, \ldots, N-1$ as:

$$\text{Expected Replacement Cost in Year } t = C_U p \left(1-p\right)^{t-1} \tag{46}$$

Here, we incur this cost only if the coin flips yielded "no fail" events in years $t = 1, 2, \ldots, t-1$, then a "fail" event in year t (i.e., we incur the cost through the geometric random variable's probability mass of having the first failure in year t, which is given by $p\left(1-p\right)^{t-1}$).

Since we assume the item must be replaced in year N if it has not already failed, the expected replacement cost does *not* depend on the result of a coin flip in year N, in the same way that replacement in year 1 precludes dependence on the year 1 coin flip. Rather, we incur this cost with certainty in year N, as long as the item had not failed in previous years $t = 1, 2, \ldots, N-1$; In other words, the expected cost is given by:

$$\text{Expected Replacement Cost in Year } N = C_P \left(1-p\right)^{N-1} \tag{47}$$

where the planned replacement cost is used.

In order to illustrate the computation of the cash flows, we further assume that, if the item is not replaced now, the plant faces a loss of revenue due to a shutdown at a cost of C_D per day. In this case, the expected downtime cost incurred in year $t = 1, 2, \ldots, N-1$ is given by:

$$\text{Expected Downtime Cost in Year } t = DC_D p \left(1-p\right)^{t-1}. \tag{48}$$

Again, we assume that the planned replacement in year $t = N$ precludes dependence on the coin flip and therefore also precludes incurring any downtime cost in that year, though we acknowledge other assumptions are possible. Thus, for practical purposes, there is no coin flip in year N.

More generally, if the item is not replaced now, the plant will face the possibility of increased costs due to reliability issues. Such costs include increased inspection costs, downtime costs for a week-long shutdown, lost revenue from a 6-hour shutdown, costs associated with the emergency replacement of an item, etc. We can express the above costs in the following functional form: $Re\_Cost\left(p, t, C_1, C_{2,\cdots}, C_M\right)$, where $C_1, C_{2,\cdots}, C_M$ are costs relevant for the considered item, and, in general, p could be a vector that incorporates multiple types of shutdown. In our case, the expected downtime cost in year $t = 1, 2, \ldots, N$ is given by a function: $Re\_Cost\left(p, t, C_P, C_U, C_D, D, N\right)$, with just a scalar value for p.

There are two relevant time-series of cash flows: one for replacing the item now and another for replacing it later, either at failure or at the horizon year N. First, consider replacing the item

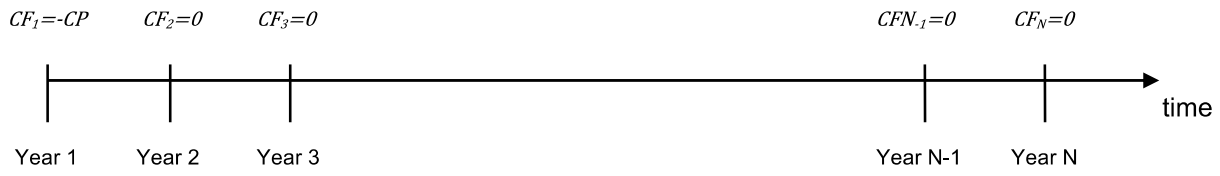


Figure 3. Graphical representation of option 1: replace now.

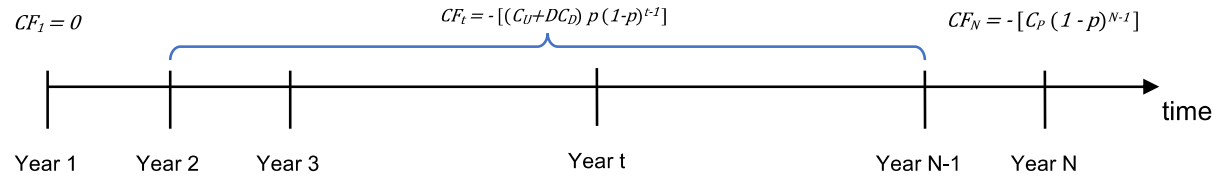


Figure 4. Graphical representation of option 2: replace later.

today; in this case, we simply incur the cost of planned replacement at time $t = 1$ (i.e., we incur cost C_P.)

The second time-series of cash flows is for replacing the item in the future. In this case, for every year t, we have the expected replacement cost and downtime cost. So, for any year $t = 1, 2, \ldots, N-1$, the cash flows will be computed as:

$$\text{Cash Flow Replaced in Year } t = -\left[(C_U + DC_D)\, p\, (1-p)^{t-1}\right] \tag{49}$$

and for the final year as:

$$\text{Cash Flow Replaced in Year } N = -\left[C_P\, (1-p)^{N-1}\right]. \tag{50}$$

Note the minus sign in front of the cash flows because all of them are costs (i.e., cash outflows). The timelines of the two options are illustrated as follows:

The NPV of option 1 is:

$$\text{NPV option 1} = -C_P \tag{51}$$

The NPV of option 2 is:

$$\text{NPV option } 2 = -\left[\sum_{t=1}^{N-1} \frac{(C_U + DC_D)\, p\,(1-p)^{t-1}}{(1+R)^{t-1}} + \frac{C_P\,(1-p)^{N-1}}{(1+R)^{N-1}}\right] \tag{52}$$

We can compare these options in two ways. The first is to simply compute the NPV as the difference between the NPV of option 1 and that of option 2 (i.e., NPV $=$ NPV option $1 -$ NPV option 2). The resulting equation is:

$$\text{NPV} = -C_P + \left[\sum_{t=1}^{N-1} \frac{(C_U + DC_D)\, p\,(1-p)^{t-1}}{(1+R)^{t-1}} + \frac{C_P\,(1-p)^{N-1}}{(1+R)^{N-1}}\right] \tag{53}$$

If the project in question is the only one under consideration, and if NPV > 0, the decision is to replace the item today; otherwise, we replace it later.

We now extend the above logic to the planned replacement occurring in year T_0. We had assumed $T_0 = 1$, but now we allow for delaying this planned replacement to a later year, albeit at the risk of incurring a failure prior to T_0, along with associated unplanned replacement and downtime costs. If the planned replacement happens at time T_0, the corresponding time-series of cash flows will become as follows: one for replacing the item either at failure before T_0 or at T_0, and another for replacing it later, either at failure or at the horizon year N.

The first time-series of cash flows is for replacing the item at T_0. In this case, for every year $t = 1, 2, \ldots, T_0 - 1$, we have the expected replacement cost and downtime cost. So, for any year $1, 2, \ldots, T_0 - 1$, the cash flows will be computed as:

$$\text{Cash Flow Replaced in Year } t = -\left[(C_U + DC_D)\, p\,(1-p)^{t-1}\right] \tag{54}$$

for $t = T_0$ the expected cash flow is:

$$\text{Cash Flow Replaced in Year } T_0 = -\left[C_P\,(1-p)^{T_0-1}\right] \tag{55}$$

The timeline of this option is illustrated as follows:

$$\text{NPV option } 1 = -\left[\sum_{t=1}^{T_0-1} \frac{(C_U + DC_D)\, p\,(1-p)^{t-1}}{(1+R)^{t-1}} + \frac{C_P\,(1-p)^{T_0-1}}{(1+R)^{T_0-1}}\right] \tag{56}$$

Note that, if $T_0 = 1$, then the first term yields zero, and the NPV of option 1 reduces to that discussed above (i.e., it equals $-C_P$.)

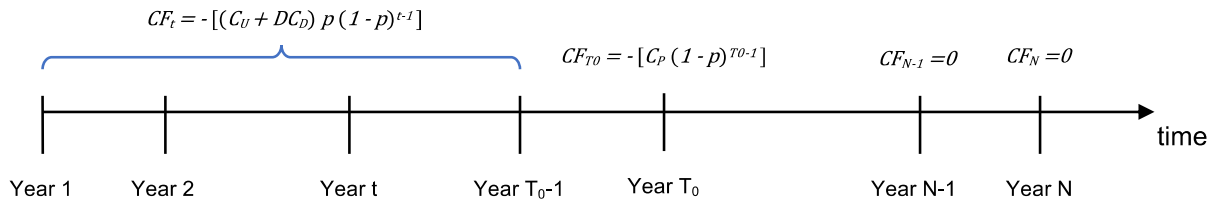


Figure 5. Graphical representation of option 1: planned replacement at T_0.

The second time-series of cash flow is for attempting to delay replacement of the item to time N, and incurring the risk of unplanned replacement and downtime in the meantime. It is the same as we calculated before.

$$\text{NPV option } 2 = -\left[\sum_{t=1}^{N-1} \frac{(C_U + DC_D)\, p\,(1-p)^{t-1}}{(1+R)^{t-1}} + \frac{C_P\,(1-p)^{N-1}}{(1+R)^{N-1}}\right] \tag{57}$$

We can compute NPV as:

$$\begin{aligned} \text{NPV} &= \text{NPV Option } 1 - \text{NPV Option } 2 && (58)\\ &= \left[\sum_{t=T_0}^{N-1} \frac{(C_U + DC_D)\, p\,(1-p)^{t-1}}{(1+R)^{t-1}} + \frac{C_P\,(1-p)^{N-1}}{(1+R)^{N-1}} - \frac{C_P\,(1-p)^{T_0-1}}{(1+R)^{T_0-1}}\right] && (59) \end{aligned}$$

A new RAVEN **External Model** with `subType` **`'LOGOS.IncrementalNPV'`** was created to compute the NPV described above.

Example RAVEN Input **`<ExternalModel>`** XML:

```
<ExternalModel name="rvi_model" subType="LOGOS.IncrementalNPV">
  <variables>fp,rvi_npv_a,rvi_npv_b</variables>
  <alias variable="rvi_p_failure" type="input">fp</alias>
  <Cp>19.82</Cp>
  <Cu>39.64</Cu>
  <fp>0.05</fp>
  <Cd>1.</Cd>
  <D>30</D>
  <options>
```

```
    <Td>1, 4</Td>
    <output>rvi_npv_a,rvi_npv_b</output>
  </options>
  <discountRate>0.03</discountRate>
  <startTime>2019</startTime>
  <lifetime>20</lifetime>
</ExternalModel>
```

In order to use this external model to compute NPV, **'LOGOS.IncrementalNPV'** should be always used for **subType**. Except for the common nodes **<variables>** and **<alias>**, this entity accepts the following sub-nodes:

- **<Cp>**, ***float, required parameter***, specifies the cost of planned replacement.
- **<Cu>**, ***float, required parameter***, specifies the cost of unplanned replacement.
- **<Cd>**, ***float, required parameter***, specifies the cost of shutdown per day.
- **<D>**, ***integer, required parameter***, specifies the number of days plant is off-line, if a shutdown occurs.
- **<fp>**, ***float, required parameter***, specifies the probability of item failure over one year.
- **<discountRate>**, ***float, required parameter***, specifies the discount rate.
- **<inflation>**, ***float, optional parameter***, specifies the inflation rate.
  Default: 0.0
- **<tax>**, ***float, optional parameter***, specifies the tax rate.
  Default: 0.0
- **<HardSavings>**, ***float, optional parameter***, specifies the hard savings that would be added to the NPV calculation.
  Default: 0.0
- **<count>**, ***integer, optional parameter***, specifies the number of the same items/components that need to be replaced at the same time.
  Default: 1.0
- **<startTime>**, ***integer, required parameter***, specifies the start time of project.
- **<lifetime>**, ***integer, required parameter***, specifies the lifetime of the project.
- **<options>**, ***optional parameter***, specifies options to compute NPVs of planned replacements occurring in different years relative to the start time.

- **<Td>**, ***comma separated integers, required parameter***, specifies the lengths of delaying the planned replacement.
- **<output>**, ***comma separated strings, required parameter***, specifies the output variables corresponding to the different specified lengths of delays in **<Td>**. In order to communicate with RAVEN, these variables need to be listed under node **<variables>**. **Note:** These variables are defined by the user and would be used to store the calculated NPVs.

The parameters **Cp, Cu, fp, Cd, inflation, and tax** can be sampled by RAVEN. If the user specifies these parameters in the input XML **<ExternalModel>**, the values for these parameters will be replaced by the sampled values from RAVEN.

Example LOGOS Incremental NPV output CSV:

```
fp,rvi_npv_a,rvi_npv_b
...
```

Note: In order to compute the NPVs, the TEAL plugin is required. Refer to `https://github.com/idaholab/raven/wiki/Plugins` for details on how to access and install the plugins.

The whole calculation flow is depicted in Fig. 6.

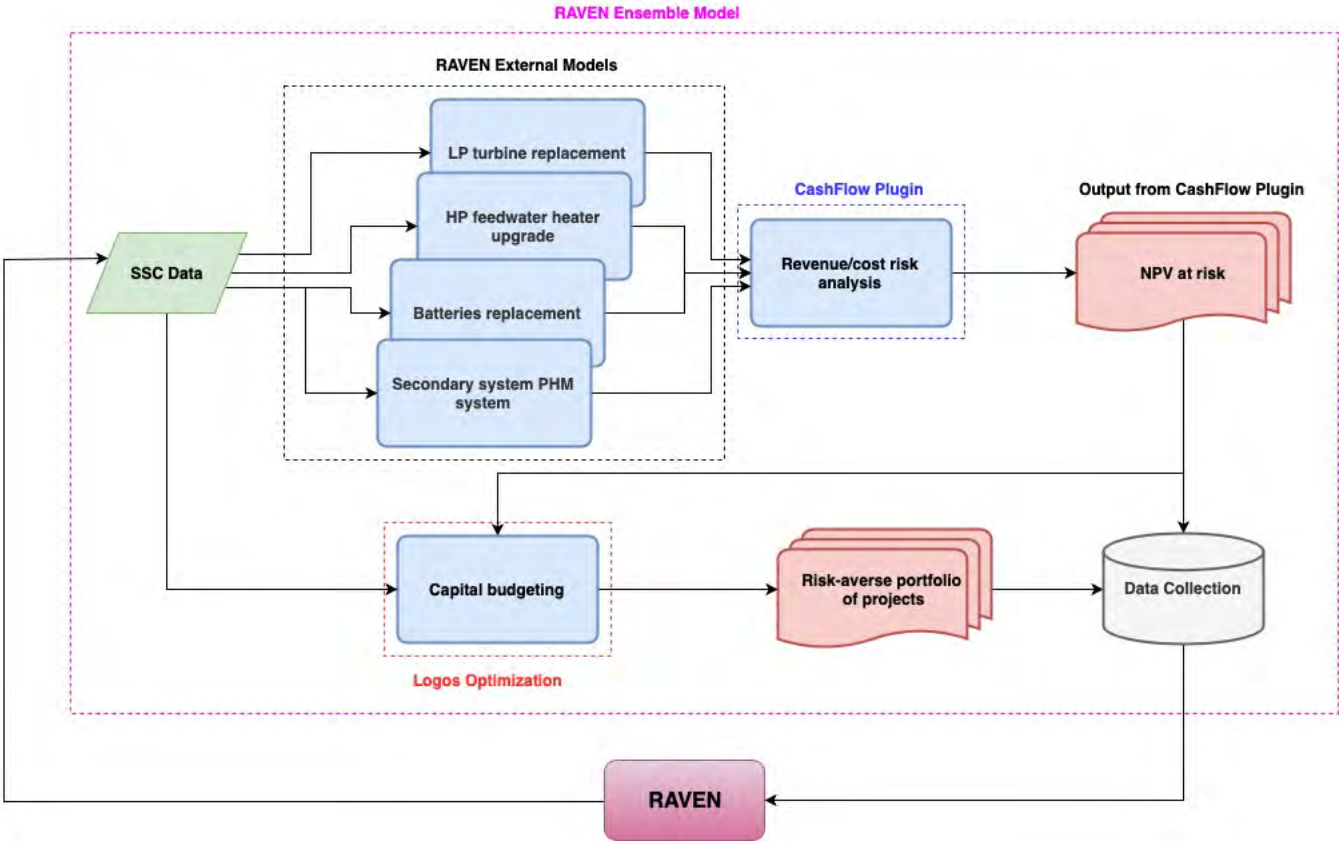


Figure 6. Risk-informed capital budgeting via RAVEN and RAVEN plugins.

9 Knapsack Models

LOGOS contains a set of Knapsack models that can be used coupled with RAVEN when the desired optimization problem requires the use of specific models to generate knapsack required parameters. More specifically, all these models would be contained in a RAVEN EnsembleModel and RAVEN optimization methods (e.g., Genetic Algorithms) would be employed to find the optimal solution.

9.1 Simple Knapsack Model

This model considers the classical Knapsack Model characterized by a set of elements that can be chosen (or not). The goal is to maximize the sum of the chosen element values provided that the sum of element cost values satisfy capacity constraint (specified in the variable defined in the **<capacity>** node).

The ID of the variables that represent cost, value, and choice of each element are indicated in the **<map>** node. The model generates two variables:

- the validity of the chosen solution (specified in the **<outcome>** node): either valid (i.e., 0), or invalid (i.e., 1) if the capacity constraint is not satisfied,
- totalValue (specified in the **<choiceValue>** node): sum of the values of the chosen elements

When calculating the **<choiceValue>** variable, if the **<capacity>** constraint is not satisfied, then the **<choiceValue>** variable is penalized by multiplying the project value by -**<penaltyFactor>**.

Example LOGOS input XML for SimpleKnapsackModel Model:

```
<ExternalModel name="knapsack"
   subType="LOGOS.SimpleKnapsackModel">
  <variables>element1Status, element2Status, element3Status,
     element4Status, element5Status,
             element1Val   ,element2Val   ,element3Val
                ,element4Val   ,element5Val,
             element1Cost  ,element2Cost  ,element3Cost
                ,element4Cost,element5Cost,
             validity,totalValue,capacityID</variables>
  <capacity>capacityID</capacity>
  <penaltyFactor>1.</penaltyFactor>
  <outcome>validity</outcome>
```

```
    <choiceValue>totalValue</choiceValue>
    <map value='element1Val'  cost='element1Cost'
      >element1Status</map>
    <map value='element2Val'  cost='element2Cost'
      >element2Status</map>
    <map value='element3Val'  cost='element3Cost'
      >element3Status</map>
    <map value='element4Val'  cost='element4Cost'
      >element4Status</map>
    <map value='element5Val'  cost='element5Cost'
      >element5Status</map>
  </ExternalModel>
```

9.2 MultipleKnapsack Model

This model considers the Multiple Knapsack Model characterized by a set of elements that can be chosen (or not) over a set of multiple knapsacks. The goal is to maximize the sum of the chosen element values provided that the sum of element cost values satisfy capacity constraints of each knapsack.

The capacity of each knapsack is defined in the **<knapsack>** node.

The ID of the variables that represent cost, value, and choice of each element are indicated in the **<map>** node. The model generates two variables:

- the validity of the chosen solution (specified in the **<outcome>** node): either valid (i.e., 0), or invalid (i.e., 1) if the capacity constraint is not satisfied,
- totalValue (specified in the **<choiceValue>** node): sum of the values of the chosen elements

When calculating the **<choiceValue>** variable, if the capacity constraints are not satisfied, then the **<choiceValue>** variable is penalized by multiplying the project value by -**<penaltyFactor>**.

Example LOGOS input XML for MultipleKnapsack Model:

```
<Models>
  <ExternalModel name="knapsack"
     subType="LOGOS.MultipleKnapsackModel">
    <variables>e1Status,e2Status,e3Status,e4Status,e5Status,
               e1Val   ,e2Val   ,e3Val   ,e4Val   ,e5Val,
               e1Cost  ,e2Cost  ,e3Cost  ,e4Cost  ,e5Cost,
```

```
                validity,totalValue,
                K1_cap,K2_cap,K3_cap</variables>
    <knapsack ID='1'>K1_cap</knapsack>
    <knapsack ID='2'>K2_cap</knapsack>
    <knapsack ID='3'>K3_cap</knapsack>
    <penaltyFactor>1.</penaltyFactor>
    <outcome>validity</outcome>
    <choiceValue>totalValue</choiceValue>
    <map value='e1Val'  cost='e1Cost' >e1Status</map>
    <map value='e2Val'  cost='e2Cost' >e2Status</map>
    <map value='e3Val'  cost='e3Cost' >e3Status</map>
    <map value='e4Val'  cost='e4Cost' >e4Status</map>
    <map value='e5Val'  cost='e5Cost' >e5Status</map>
  </ExternalModel>
</Models>
```

Document Version Information

This document has been compiled using the following version of the plugin Git repository:
dea9e0974d2543c30319046c402a2c7f73c40177 mandd Fri, 9 Jul 2021 16:51:39 -0600

References

[1] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, and J. Chen, "Raven user manual," Tech. Rep. INL/EXT-15-34123, March 2017.

[2] A. Alfonsi, C. Rabiti, D. Mandelli, J. Cogliati, C. Wang, P. W. Talbot, D. P. Maljovec, and C. Smith, "Raven theory manual and user guide," tech. rep., Idaho National Laboratory (INL), 2017.

[3] A. Koç, D. P. Morton, E. Popova, S. M. Hess, E. Kee, and D. Richards, "Prioritizing project selection," *The Engineering Economist*, vol. 54, no. 4, pp. 267–297, 2009.

[4] P. Artzner, F. Delbaen, J. M. Eber, and D. Heath, "Thinking coherently," *Risk*, vol. 10, pp. 68–71, 1997.

[5] P. Artzner, F. Delbaen, J. M. Eber, and D. Heath, "Coherent measures of risk," *Mathematical Finance*, vol. 9, no. 3, pp. 203–228, 1999.

[6] R. T. Rockafellar and S. P. Uryasev, "Optimization of conditional value at risk," *Journal of Risk*, vol. 2, pp. 21–42, 2000.

[7] G. C. Pflug, "Some remarks on the value at risk and the conditional value at risk," *in Probabilistic Constrained Optimization: Methodology and Applications*, pp. 278–287, 2000.

APPENDIX C – VERT USER MANUAL

MANUAL

INL/EXT-21-63022
Revision 0
Printed July 13, 2021

VERT User Manual

J. Condie, J. Miller, S. Ercanbrack, C. Pope

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

INL/EXT-21-63022
Revision 0
Printed July 13, 2021

VERT User Manual

Jason Condie
Idaho State University
condjaso@isu.edu

Jaden Miller
Idaho State University
milljad2@isu.edu

Spencer Ercanbrack
Idaho State University
ercaspen@isu.edu

Chad Pope
Idaho State University
popechad@isu.edu

Contents

1 Terms and Acronyms

BWR boiling water reactor

CAFTA computer-assisted fault tree analysis

GRA generation risk assessment

INL Idaho National Laboratory

LWR light water reactor

NERC-GADS North American Electric Reliability Corporation - Generating Availability Data System

SAPHIRE systems Analysis Program for Hands-On Integrated Reliability Evaluations

NPP nuclear power plant

PWR pressurized water reactor

RAVEN Risk Analysis Virtual ENvironment

SSCs systems, structures, and components

VERT Versatile Economic Risk Tool

2 Introduction

The Versatile Economic Risk Tool (VERT) is a program capable of performing economic risk analyses on nuclear power plants (NPPs). VERT utilizes and builds on generation risk assessment (GRA) methodologies to quickly and effectively estimate lost revenue caused by the unavailability of systems, structures, and components (SSCs) in a power plant. Idaho State University and Idaho National Laboratory (INL) are the drivers behind the continuous development of VERT. VERT has been implemented for several proof-of-concept case studies. A parametric study, taking data from 1980 to 2020 on plant component unavailability from the North American Electric Reliability Corporation's Generating Availability DATA System (NERC-GADS), was conducted to evaluate a generic NPP's performance.

VERT uses the Systems Analysis Program for Hands-on Integrated Reliability Evaluations (SAPHIRE) program from INL as the framework for the event and fault tree analyses. SAPHIRE is a probabilistic risk analysis (PRA) software tool that gives the user the ability to create and solve fault trees and event trees using a graphical interface. The INL began development of SAPHIRE in the mid-1980s, when the enormous potential of computer applications and software started being recognized.

VERT uses the Risk Analysis Virtual ENvironment (RAVEN) to introduce parametric variability based on the model developed in SAPHIRE. RAVEN is a software framework able to perform parametric and stochastic analyses based on the response of complex system codes. The initial development was aimed at providing dynamic risk analysis capabilities to the thermohydraulic code RELAP-7, currently under development at INL. Although the initial goal has been fully accomplished, RAVEN is now a multi-purpose stochastic and uncertainty quantification platform, capable of communicating with any system code. [1, 2]

This user manual will cover the installation and use of SAPHIRE, RAVEN, and VERT. The manual will cover all aspects necessary to use VERT to perform an economic analysis for a NPP model.

3 Installation

3.1 Overview

VERT requires the installation of both SAPHIRE and RAVEN to run. It also requires the installation of Python to be able to alter any of the code. RAVEN uses Miniconda to install the necessary Python libraries. RAVEN will also need the use of Git SCM and a C compiler. The installation of RAVEN will follow the path of:

1. Installing prerequisites (Git, conda, and C compilers)
2. Clone RAVEN from Github
3. Compile and build RAVEN

For a windows intallation, this guide will follow the RAVENWiki found at:

`https://github.com/idaholab/raven/wiki/installationWindowsNew`

For any other operating system please follow the link below and select the appropriate option:

`https://github.com/idaholab/raven/wiki/installationMain` [2]

Miniconda is a free minimal installer for conda. It is a small, bootsrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib, and a few others. [3]

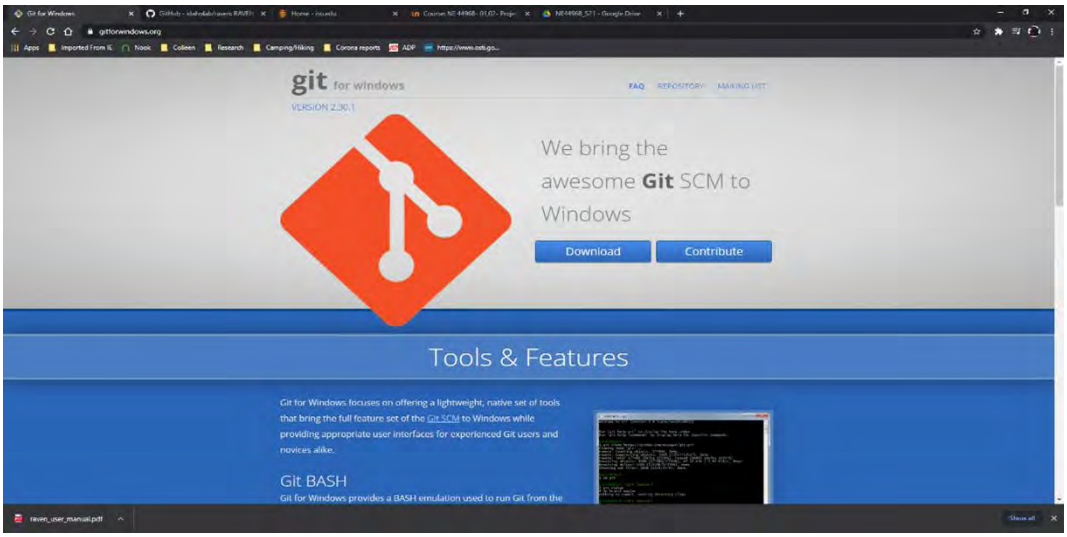
A compiler is a program that processes statements written in a particular programming language and turns them into machine language or ”code” that a computer’s processor uses.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects.

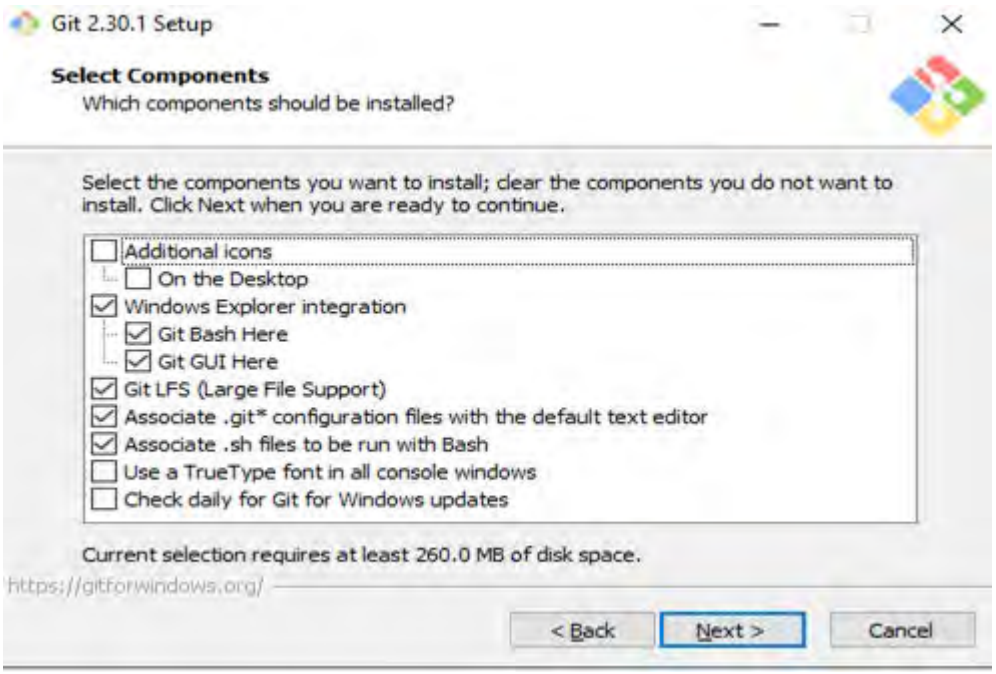
3.2 RAVEN

3.2.1 Git SCM

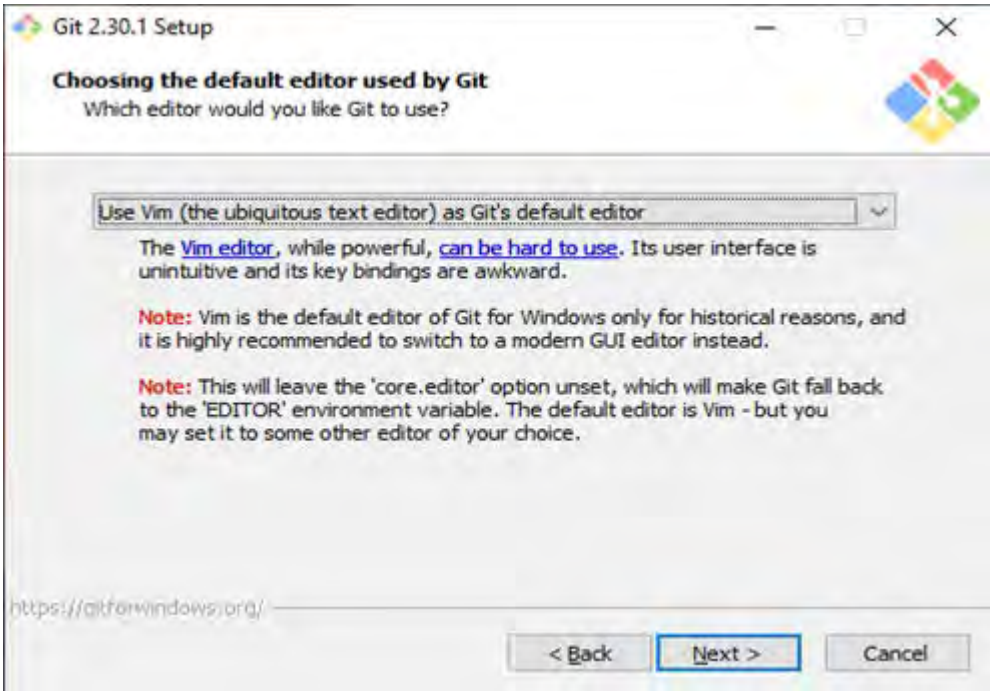
Get the latest Git SCM for windows from `https://gitforwindows.org/` by clicking on the ”Download” button.



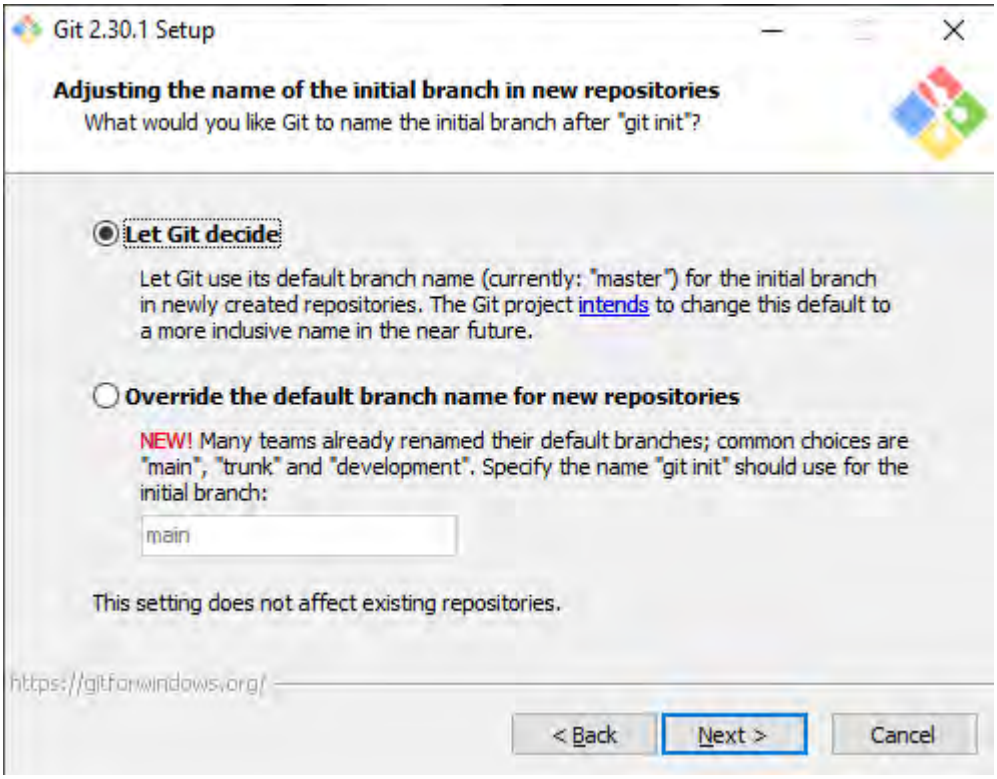
The following components must be installed.



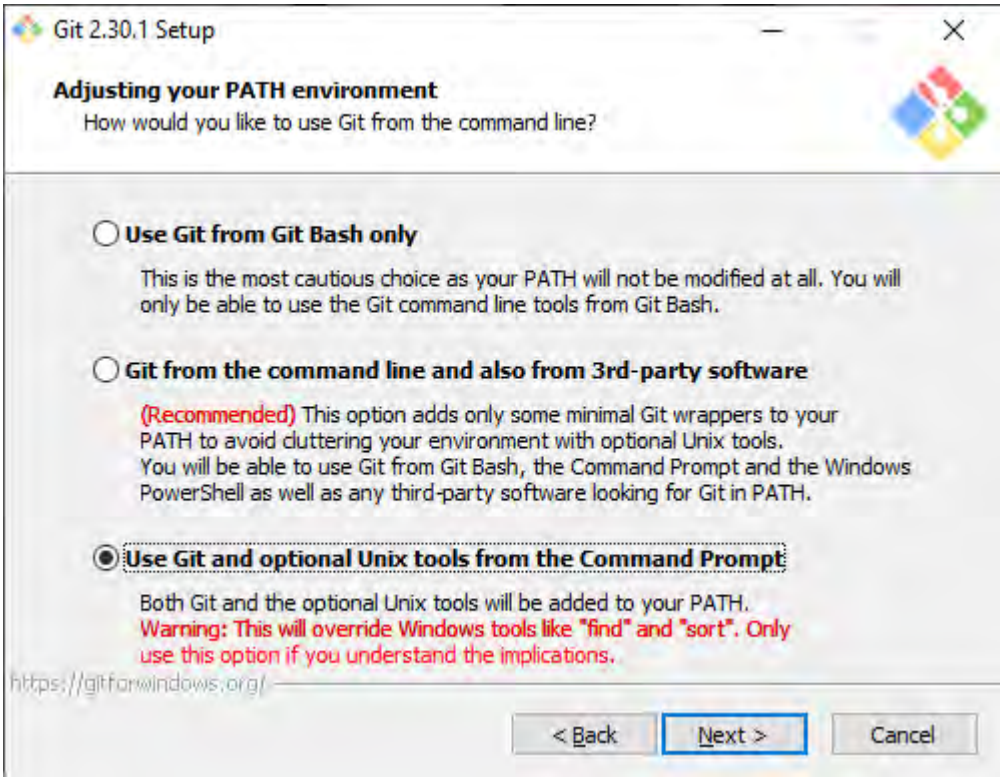
Select ”Use Vim as the default editor.”



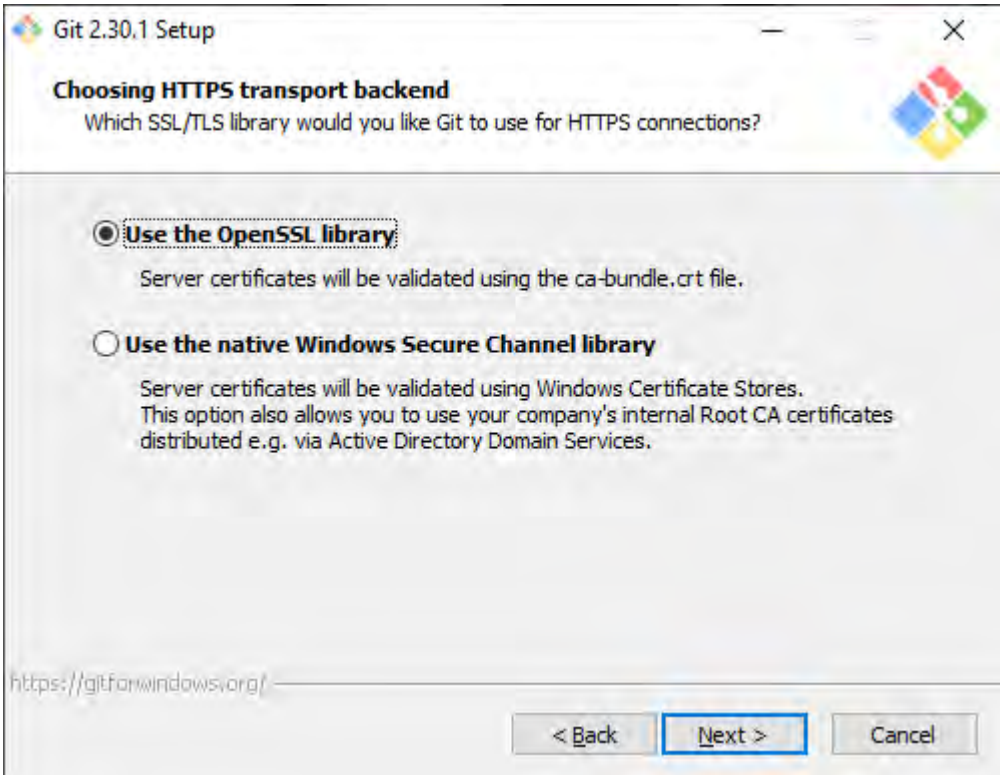
Select ”Let Git decide” for the default branch name.



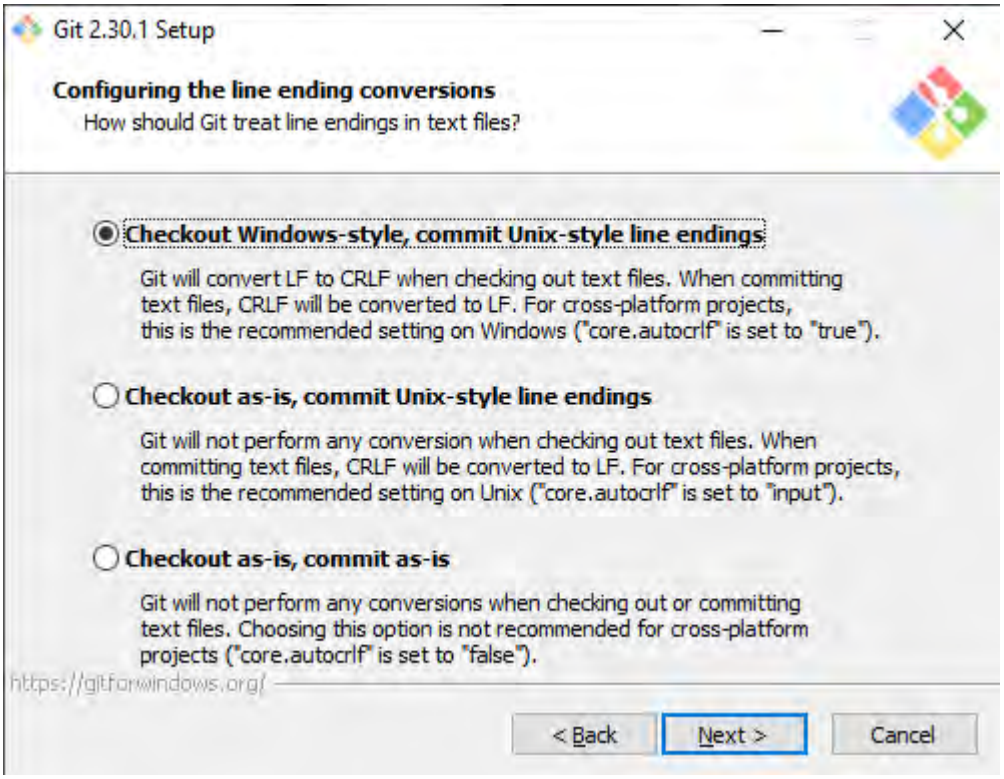
Select ”Use Git and optional Unix tools from the Command Prompt.”



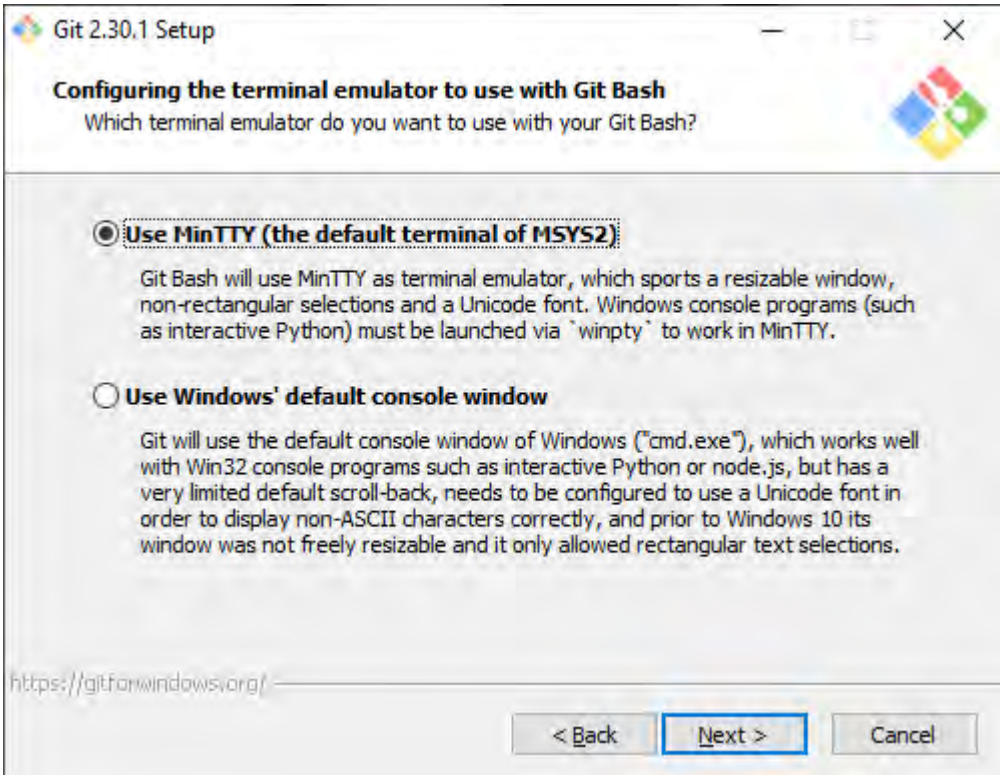
Select ”Use the OpenSSL library.”



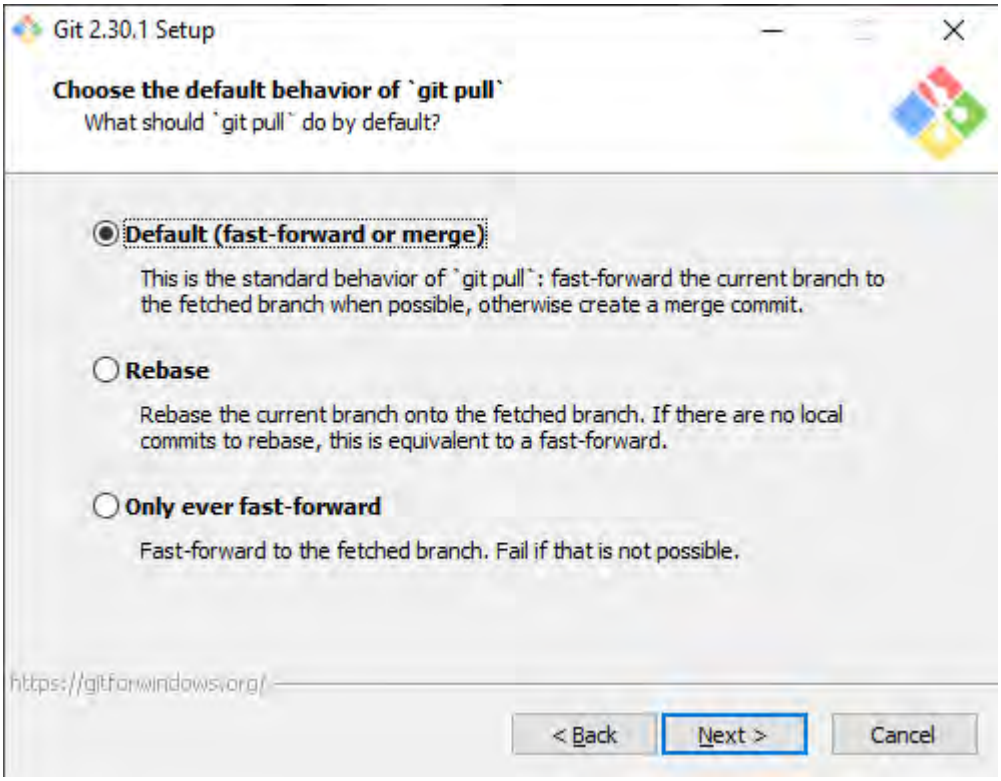
Select ”Check out Windows-style, commit Unix-style line endings.”



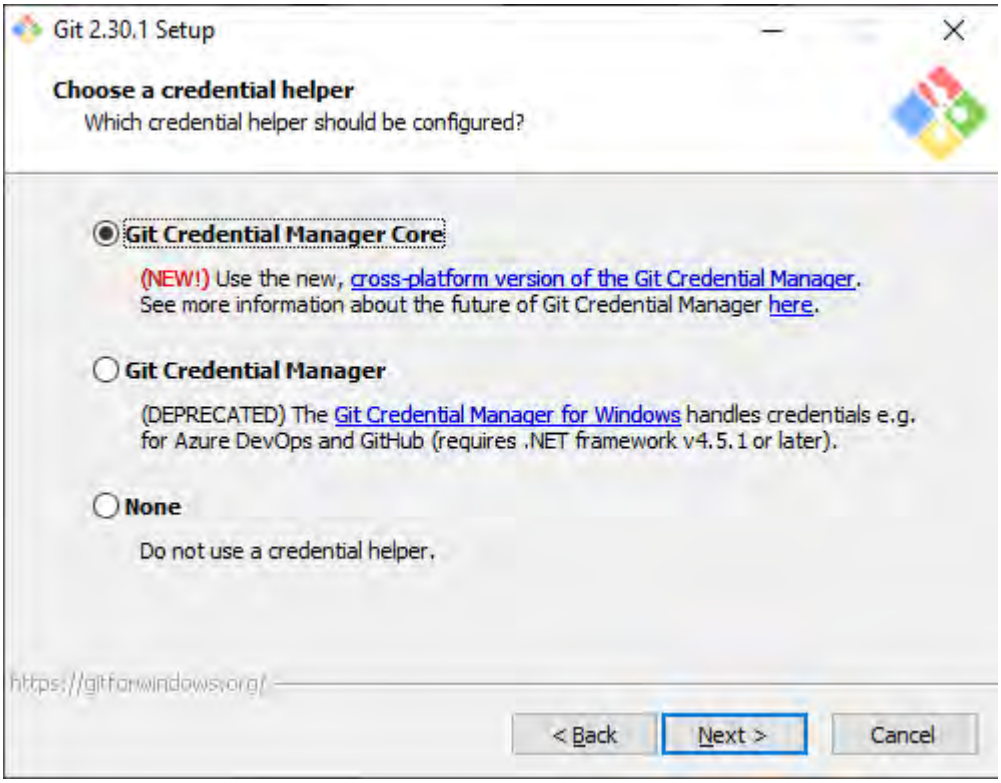
Select ”Use MinTTY (The default terminal of MSYS2).”



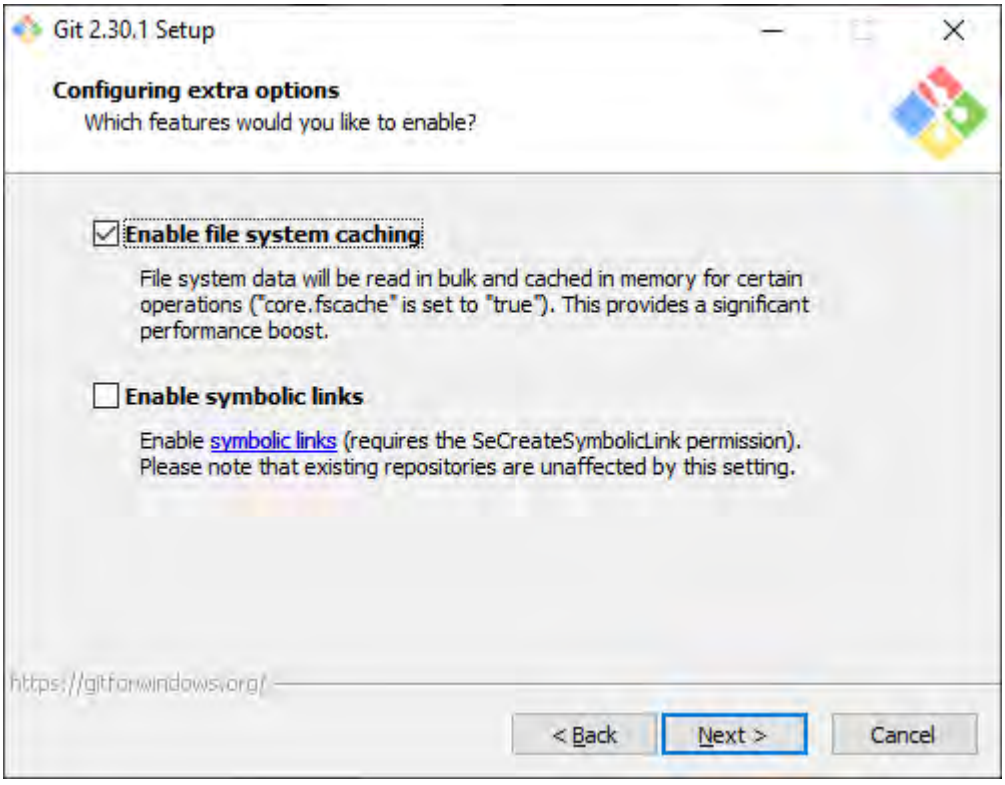
Select ”Default (fast-forward or merge).”



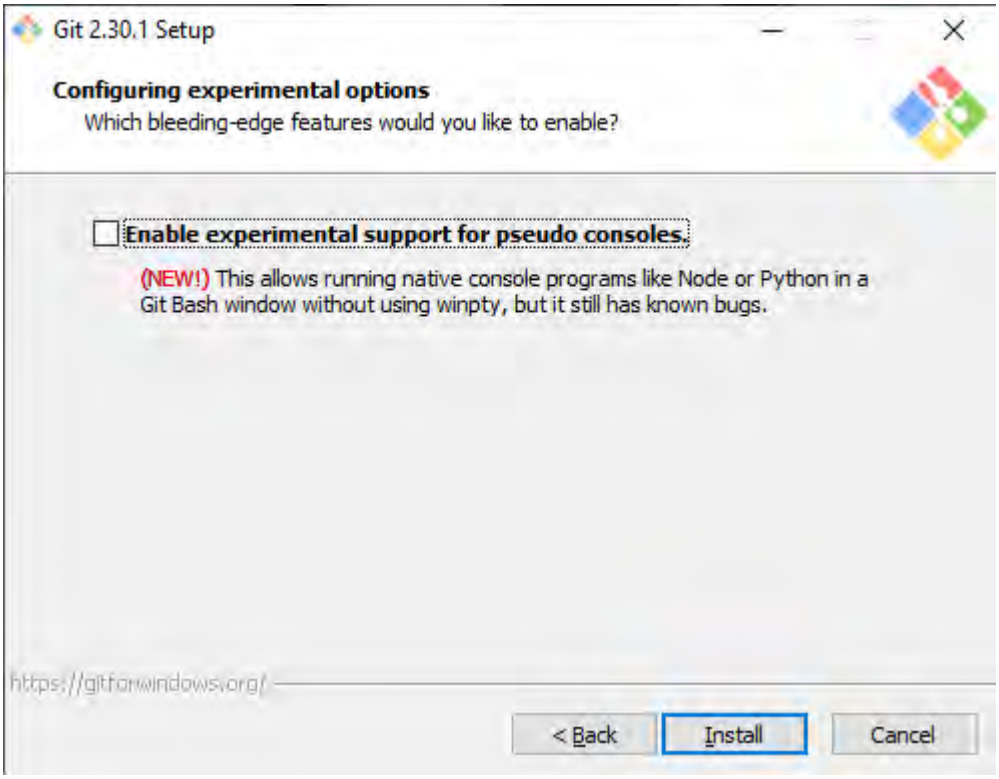
Select ”Git Credential Manager Core.”



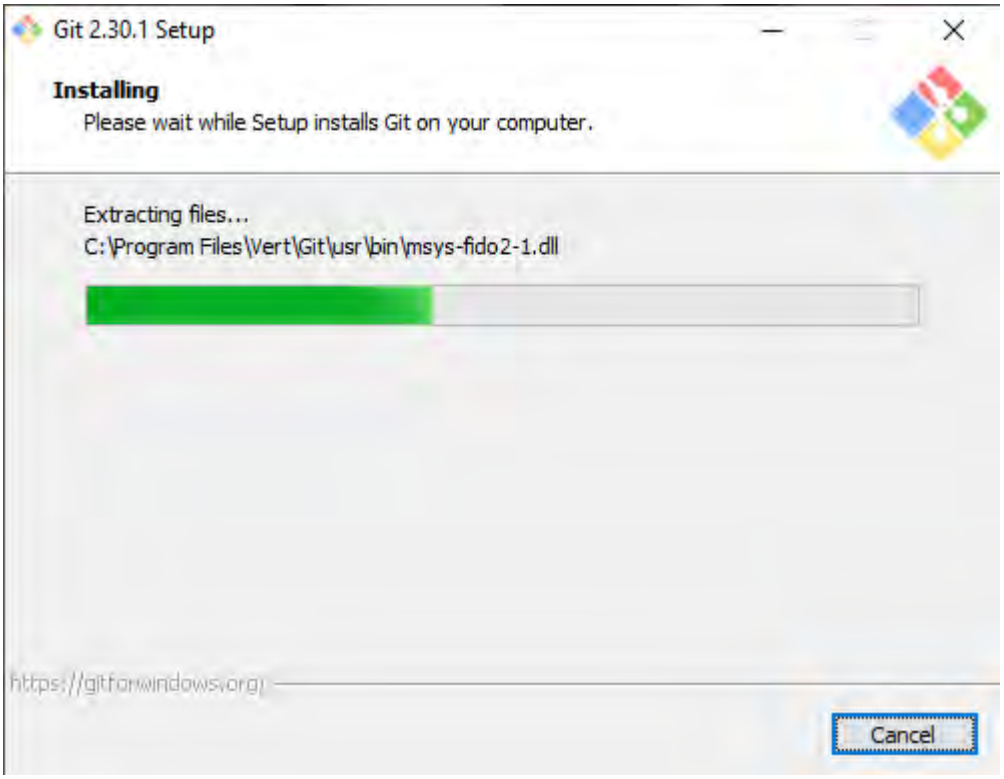
Only check ”Enable file system caching.”



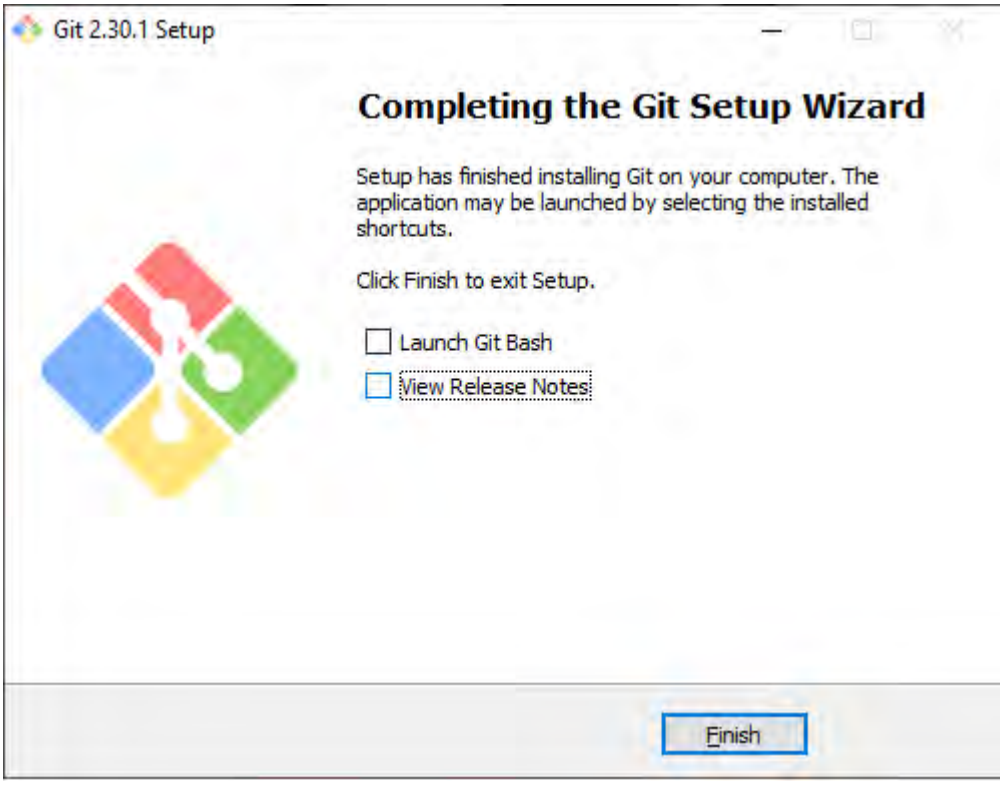
Do not check ”Enable experimental support for pseudo consoles” and click ”Install.”



Installing screen for verification.

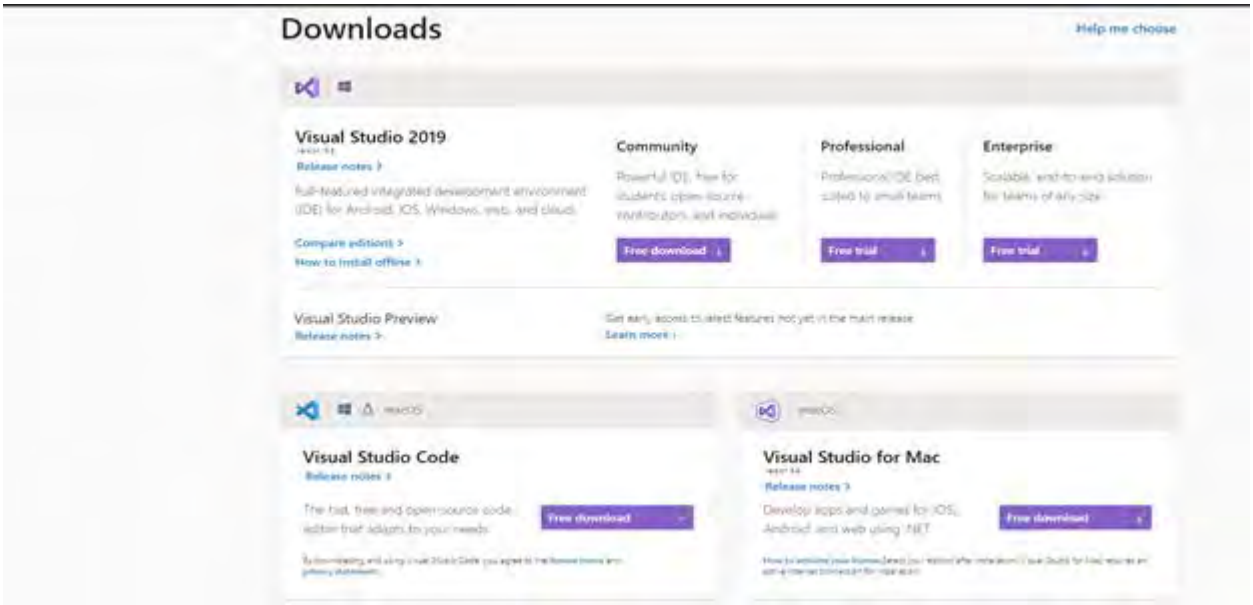


No need to look at release notes or launch now. Click ”Finish.”

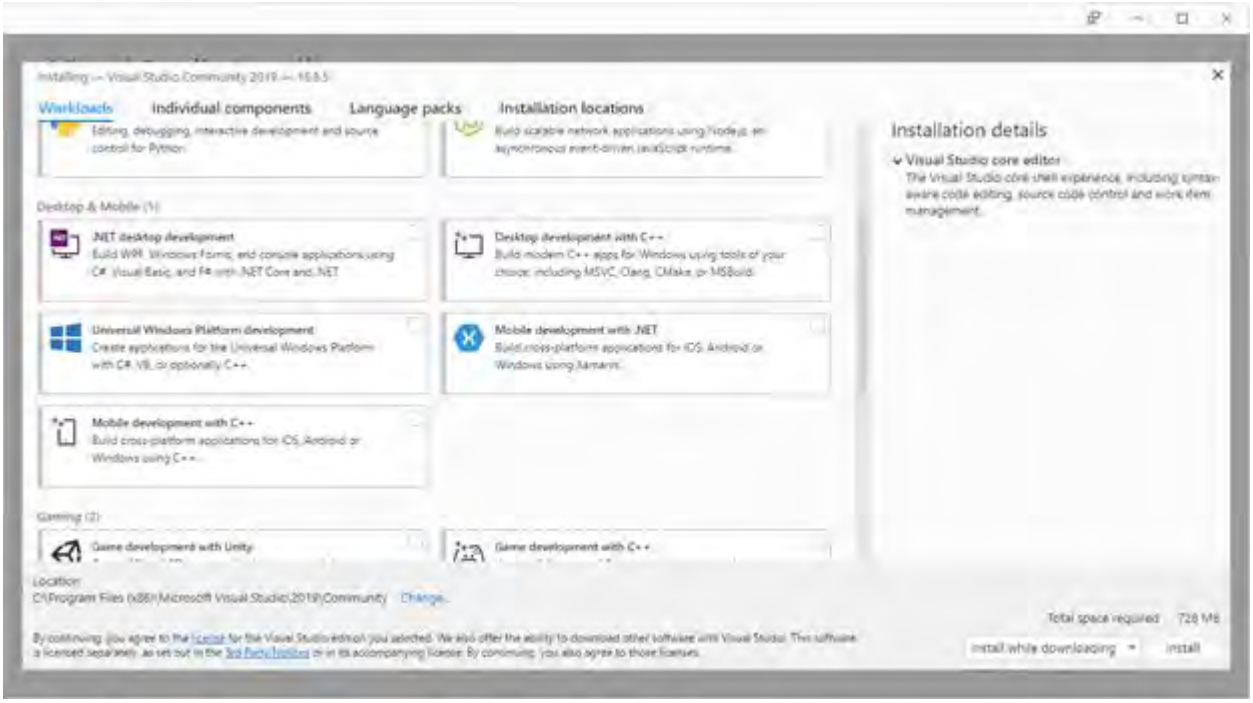


3.2.2 C compiler

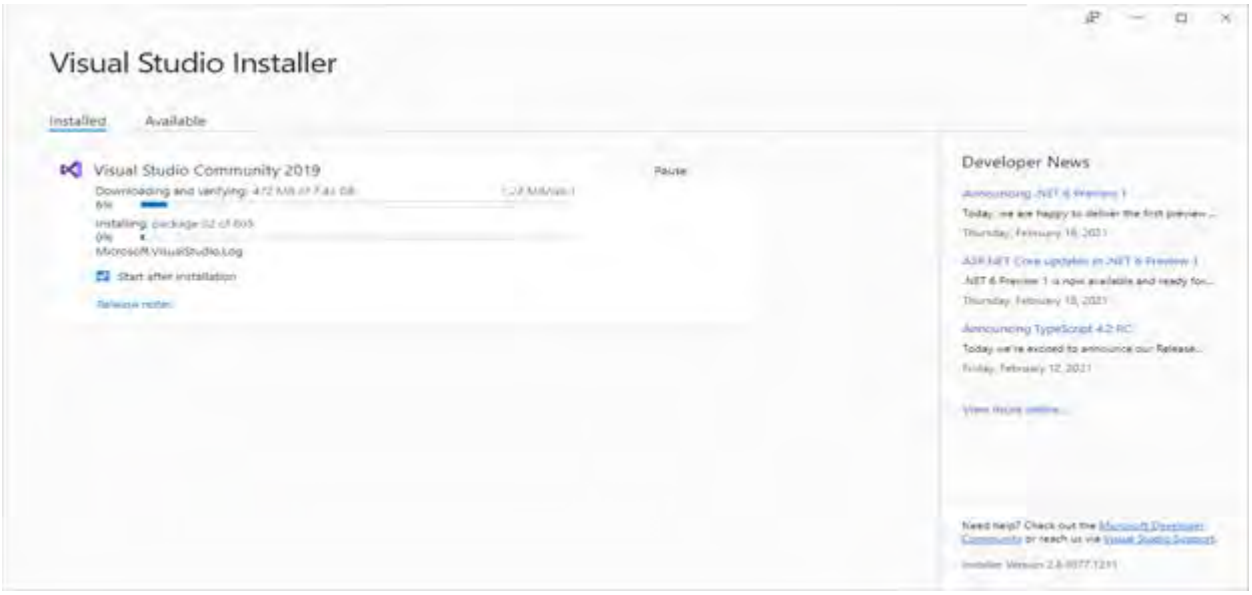
For the C compiler, this manual will use Microsoft's Visual Studio Community Edition. Go to `https://visualstudio.microsoft.com/downloads/` to download the community edition.



After downloading and running the Visual Studio installer, it will ask what features to install. For building RAVEN, "Desktop development with C++" is needed at a minimum [2, 3].

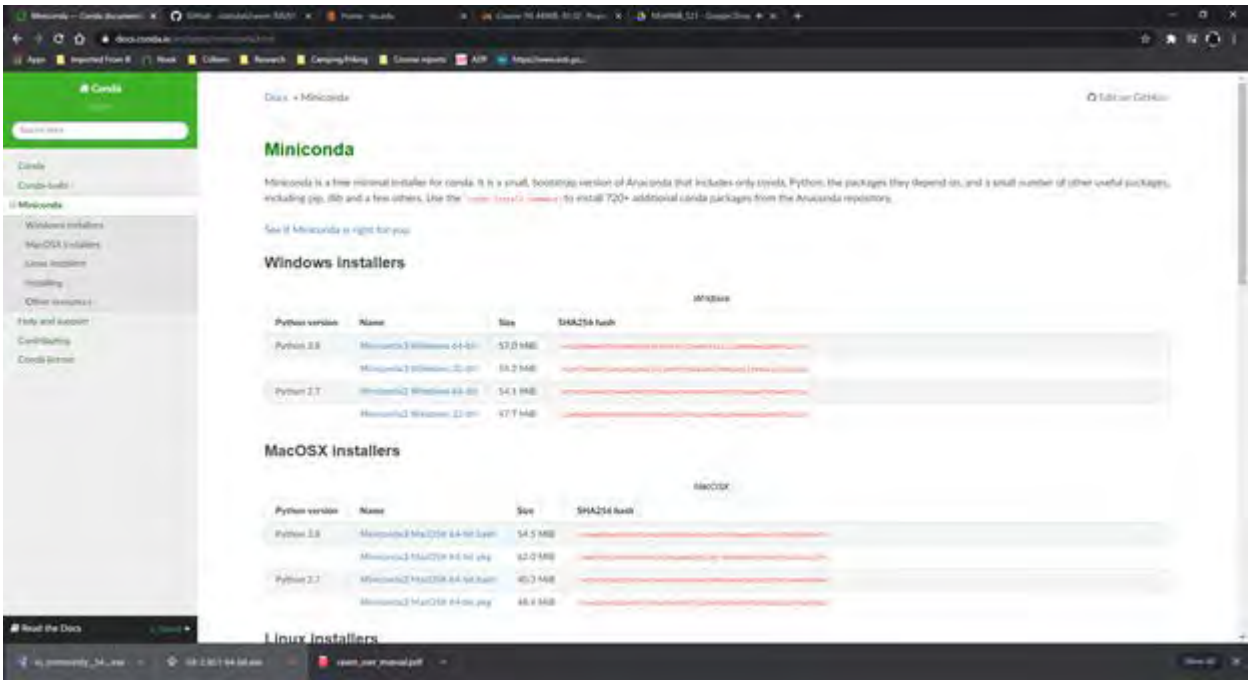


Once done, the computer will need a restart.

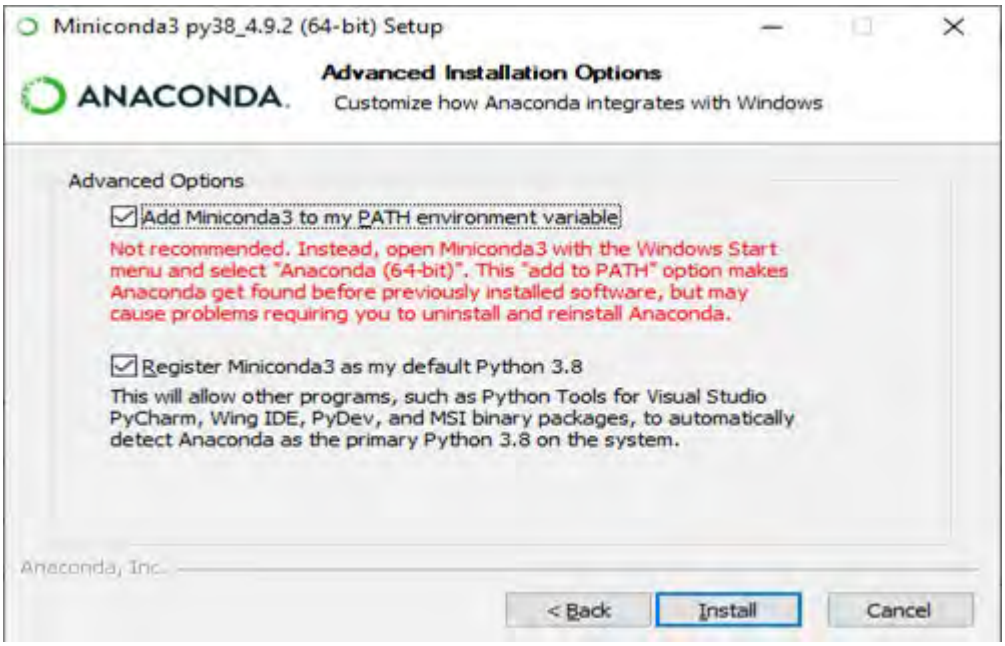


3.2.3 Conda

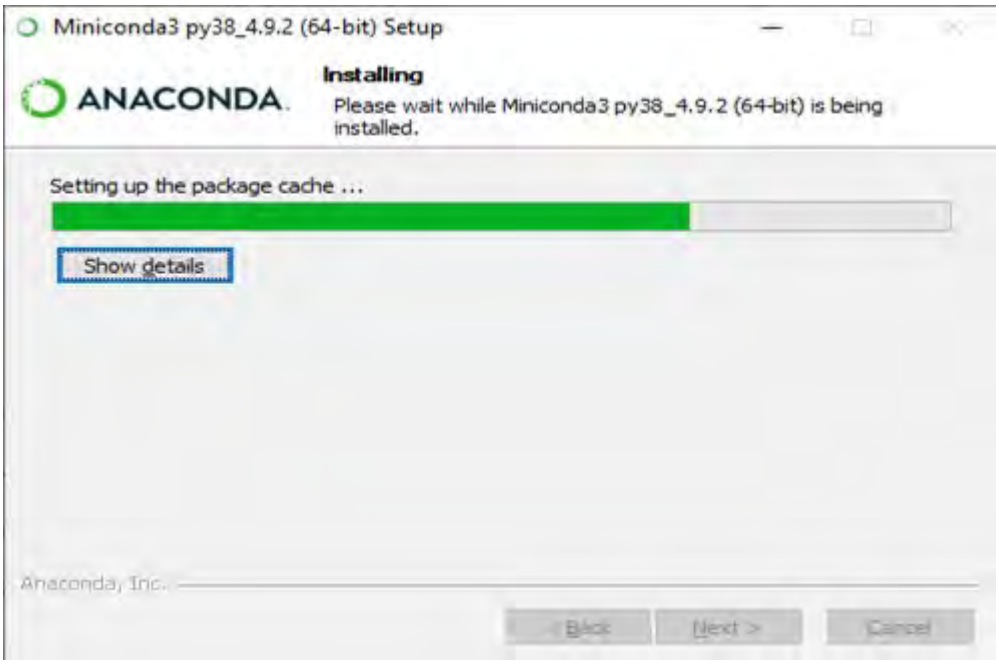
Go to `https://conda.io/miniconda.html` to download Python 3.8 - It is important to VERT that this is installed in the root directory or C: drive.



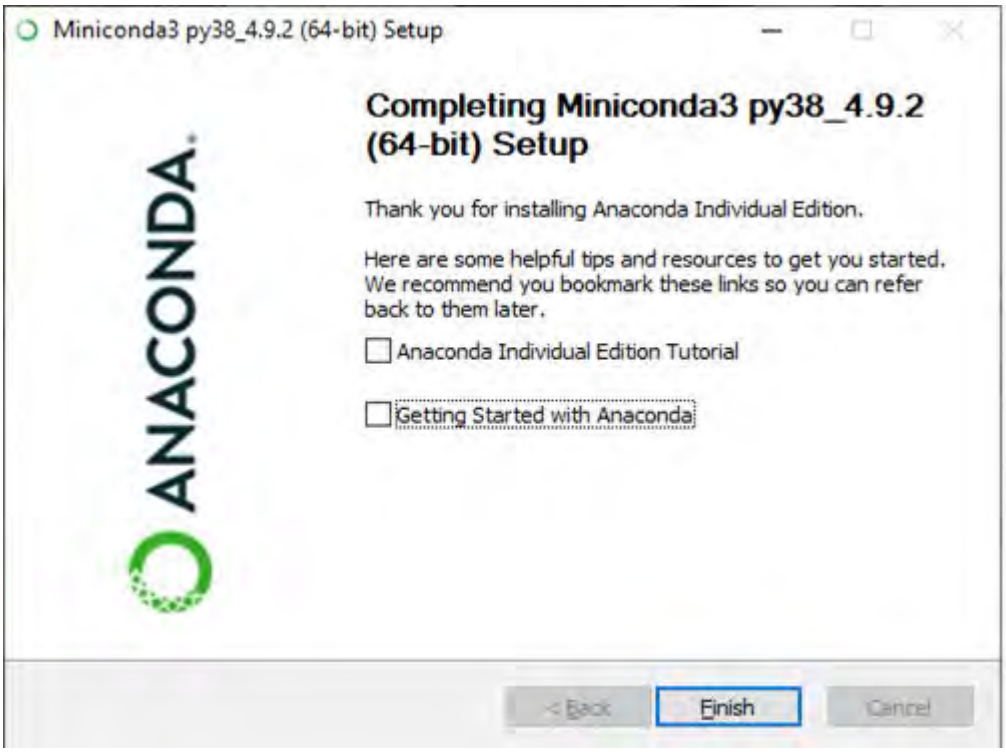
In advanced options, check ”Add Miniconda3 to my PATH environment variable,” then click ”Install.”



Installation screen for verification.

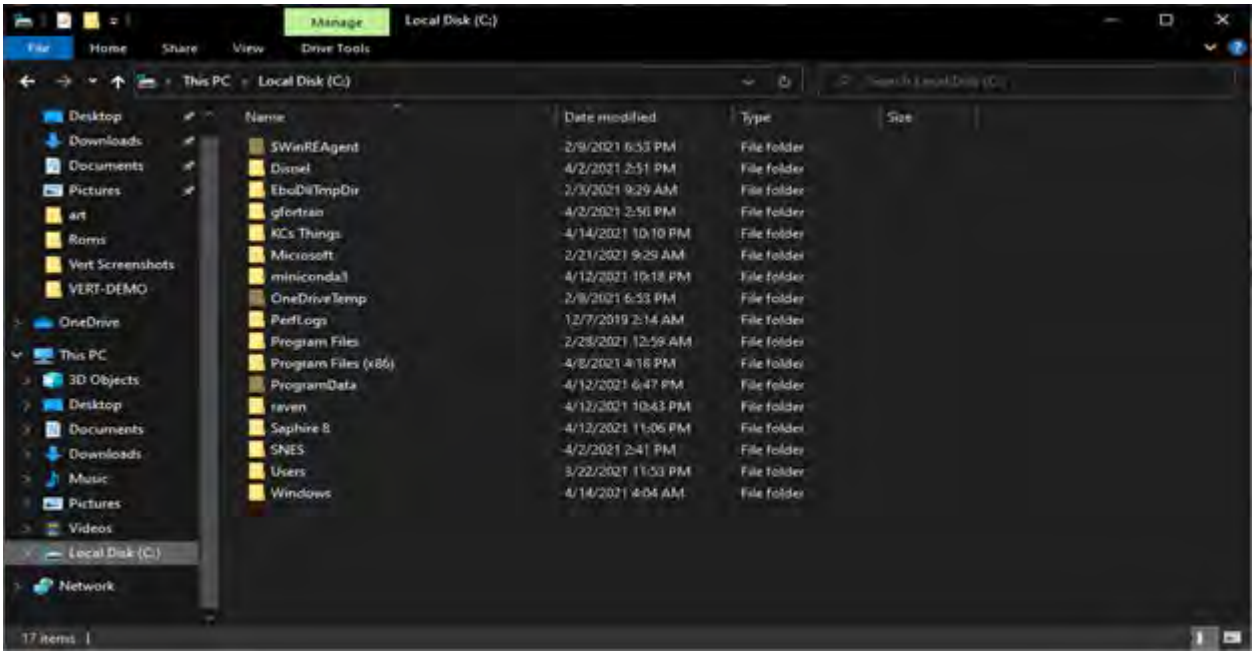


No need to follow the links, so uncheck both ”Anaconda Individual Edition Tutorial” and ”Getting Started with Anaconda,” then click ”Finish.”



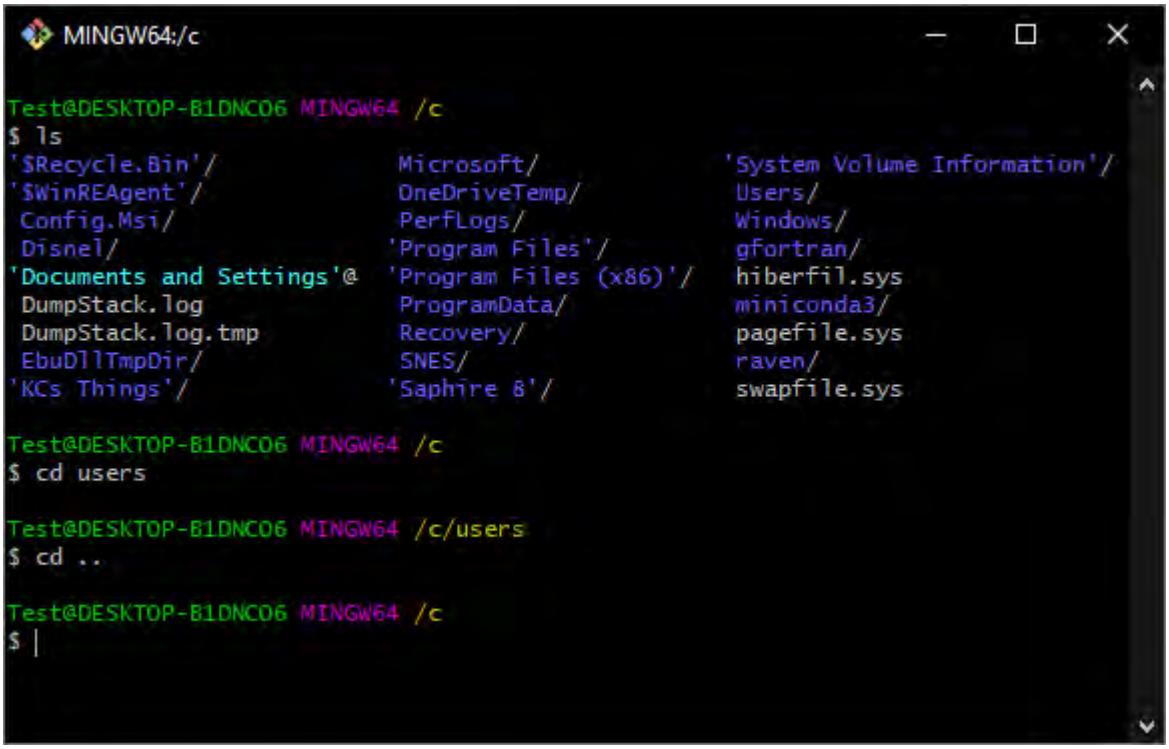
3.2.4 RAVEN

Choose the root drive or C: to install RAVEN. This should be in the same spot as the Miniconda3 folder.

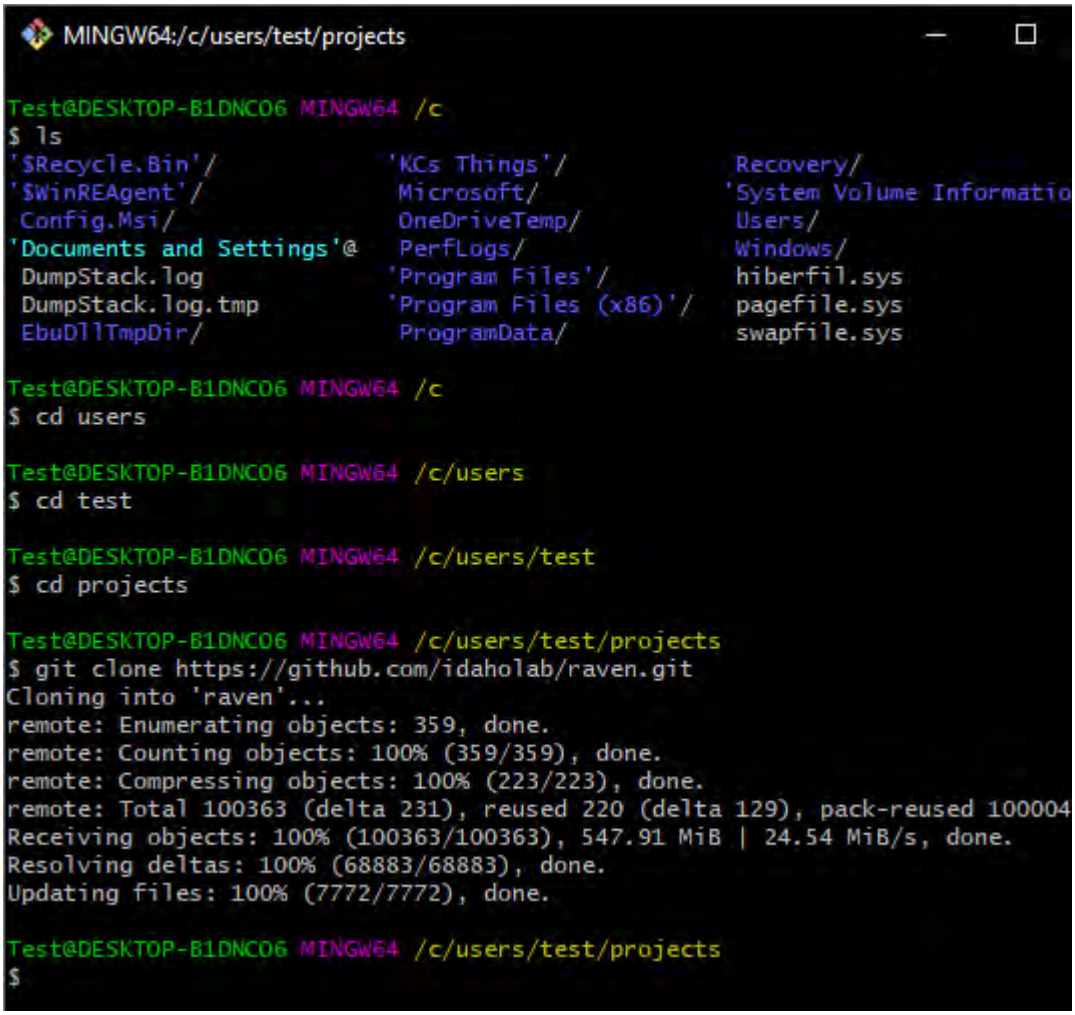


Open the GitBash terminal and use command ”cd” to get into the folder directory where you want to install RAVEN (e.g., C:).

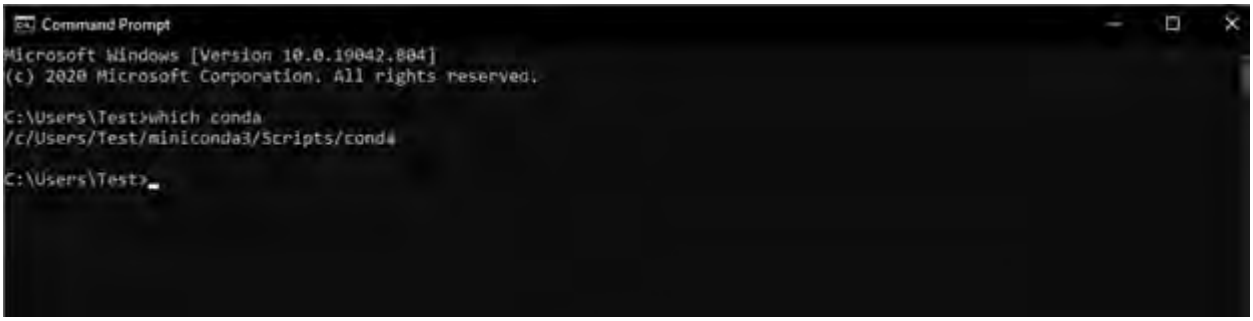
In the GitBash terminal, go to the file pathway for your projects. To see the current path, use ”pwd.” Use the command ”ls” to see the contents of the directory you are currently in. Then, use ”cd/c” to enter into the root directory. If you need to go back to the other directories, use command ”cd.”



Once in the correct directory, enter "git clone `https://github.com/idaholab/raven.git`". This will clone RAVEN from INL's RAVEN GitHub repository. [4]



To install the Python libraries, first open the windows command prompt. Find where your conda was installed by typing ”which conda” to get the pathway result.

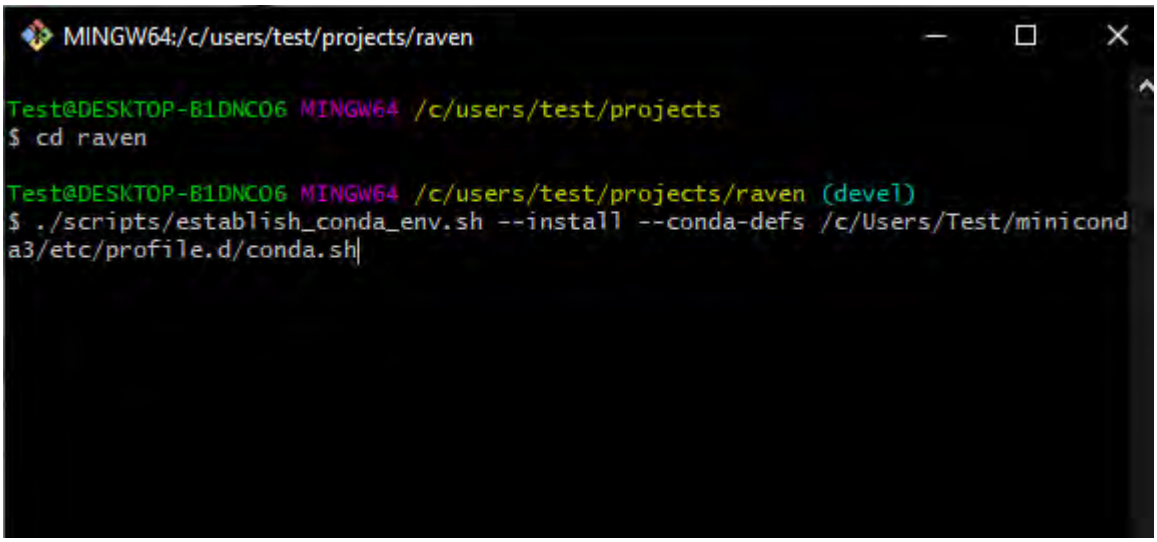


Your exact path will be slightly different depending on how conda was installed and possibly your user-name. The result above is simply **an example**. Whatever your result is, copy it and

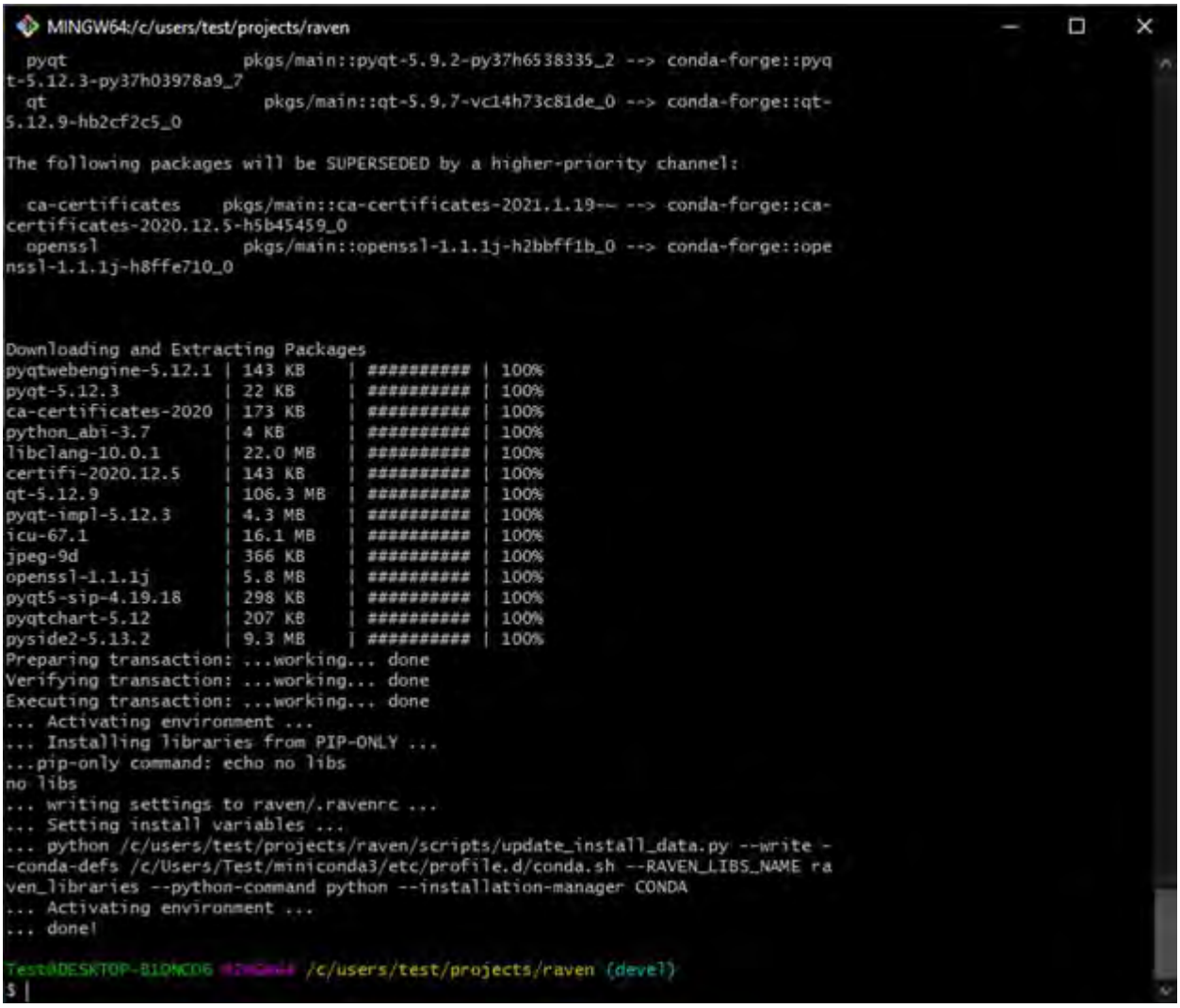
modify it as follows:

1. Remove the last two entries (the part after miniconda3)
    - Example: /c/Users/Test/miniconda3/
2. Add /etc/profile.d/conda.sh
    - Example: /c/Users/Test/miniconda3/etc/profile.d/conda.sh
      This is **the location of your conda definitions** used during the next step.

In GitBash, enter your projects pathway. Type ”cd raven”. Then, type: ”./scripts/establish_conda_env.sh −−install −−conda-defs <yourCondaDefsLocation>” with your conda pathway replacing ”<yourCondaDefsLocation>”. [4]

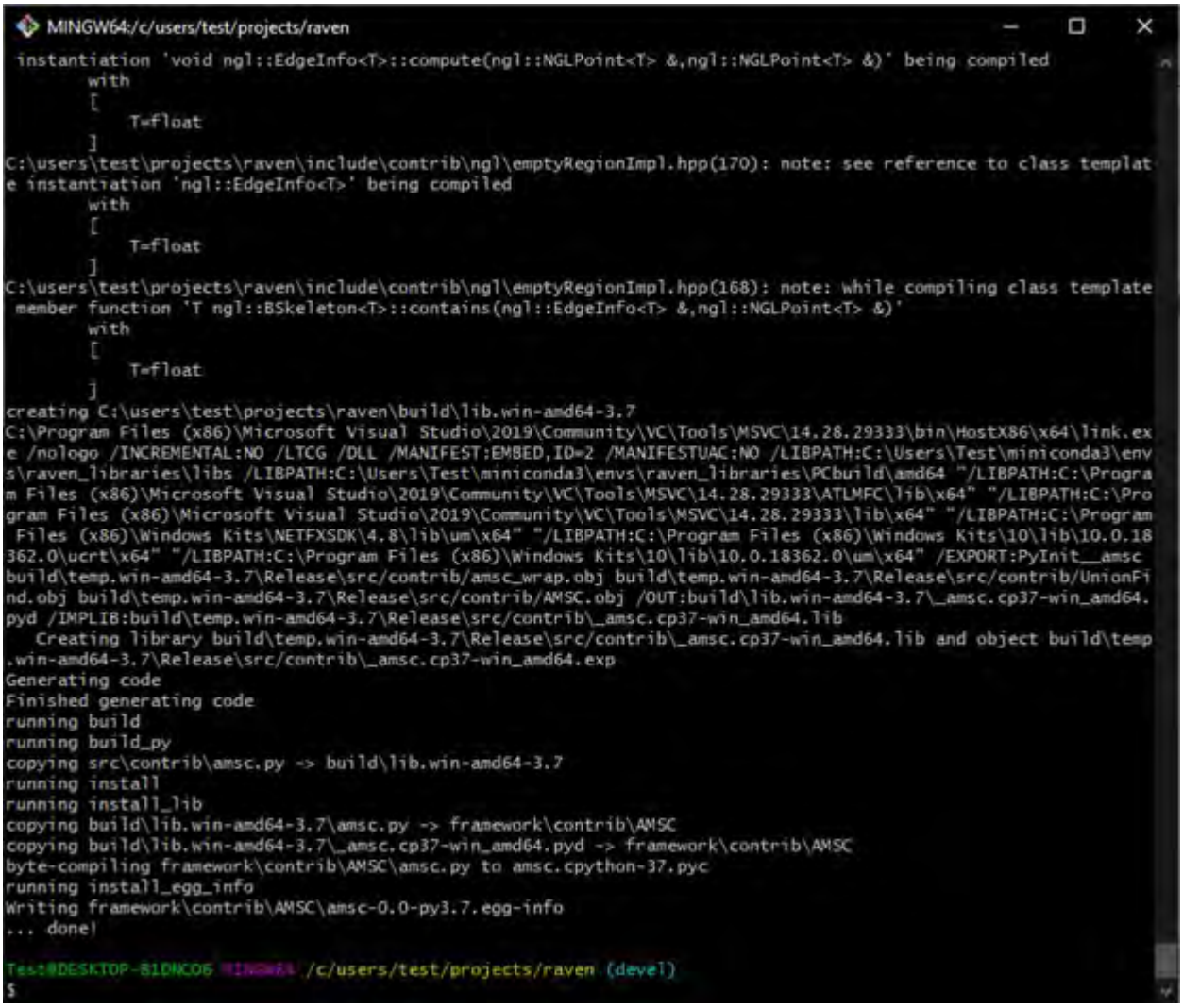


This step installs the libraries to the RAVEN pathway. The completed screen for verification shown below. If you can not install the libraries, see Installing RAVEN Libraries. [4]

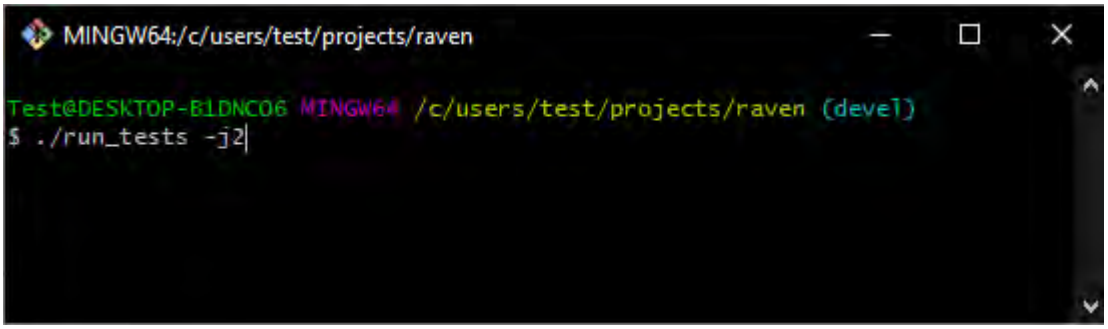


Compiling RAVEN:
In GitBash under the RAVEN pathway, type in "./build_raven".

Completion screen for verification.

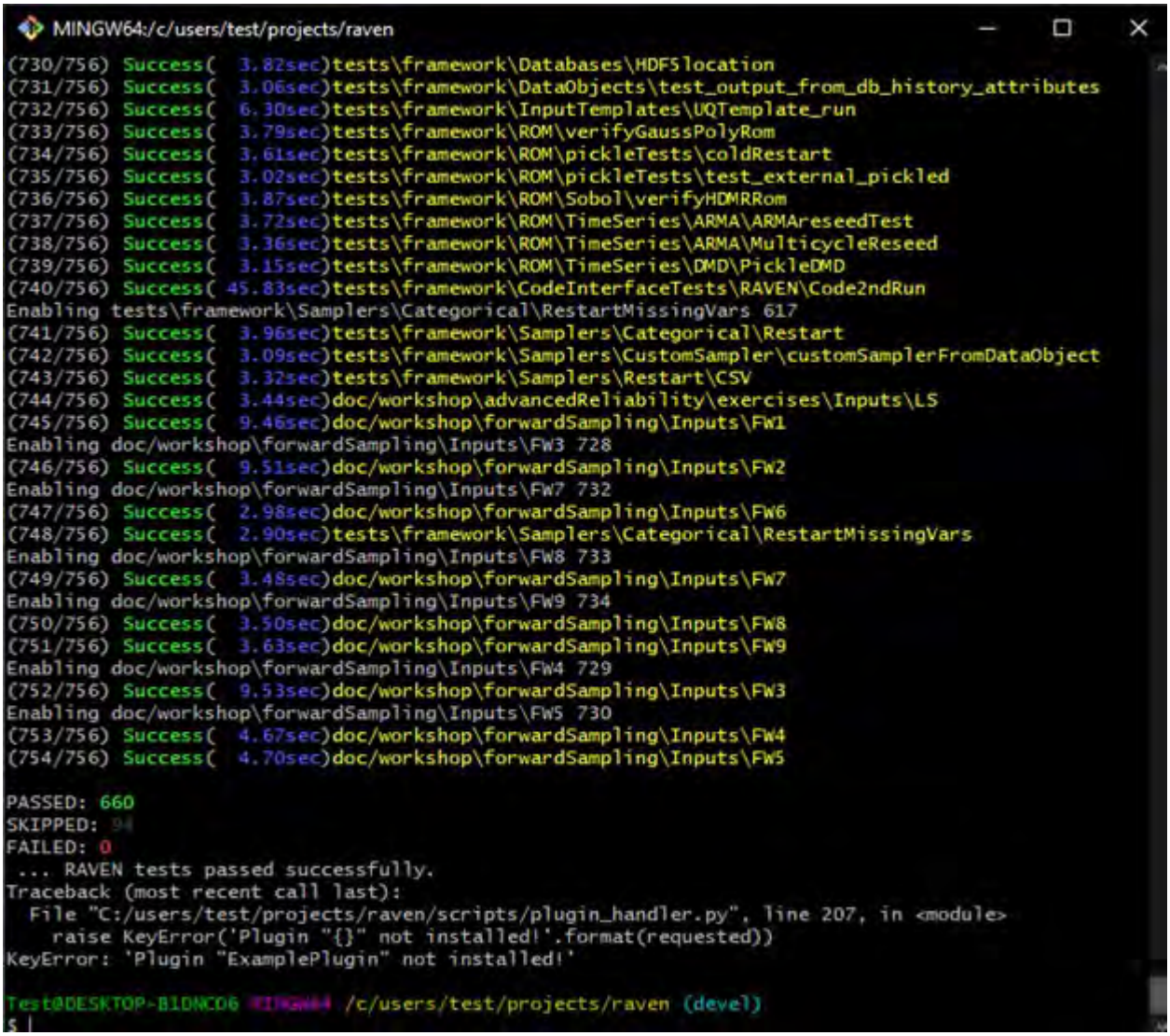


Test the RAVEN installation by typing "./run_tests -j2" under the RAVEN pathway in the Git-Bash terminal.



Completion screen for verification. Most of the tests should pass, several may skip, but few to

none should fail.



Go to the file explorer, and there will be a RAVEN folder in the projects directory. In the RAVEN folder, double click on ”raven_framework.bat” to run RAVEN, producing the following screen:

This completes the installation of RAVEN.

3.3 SAPHIRE

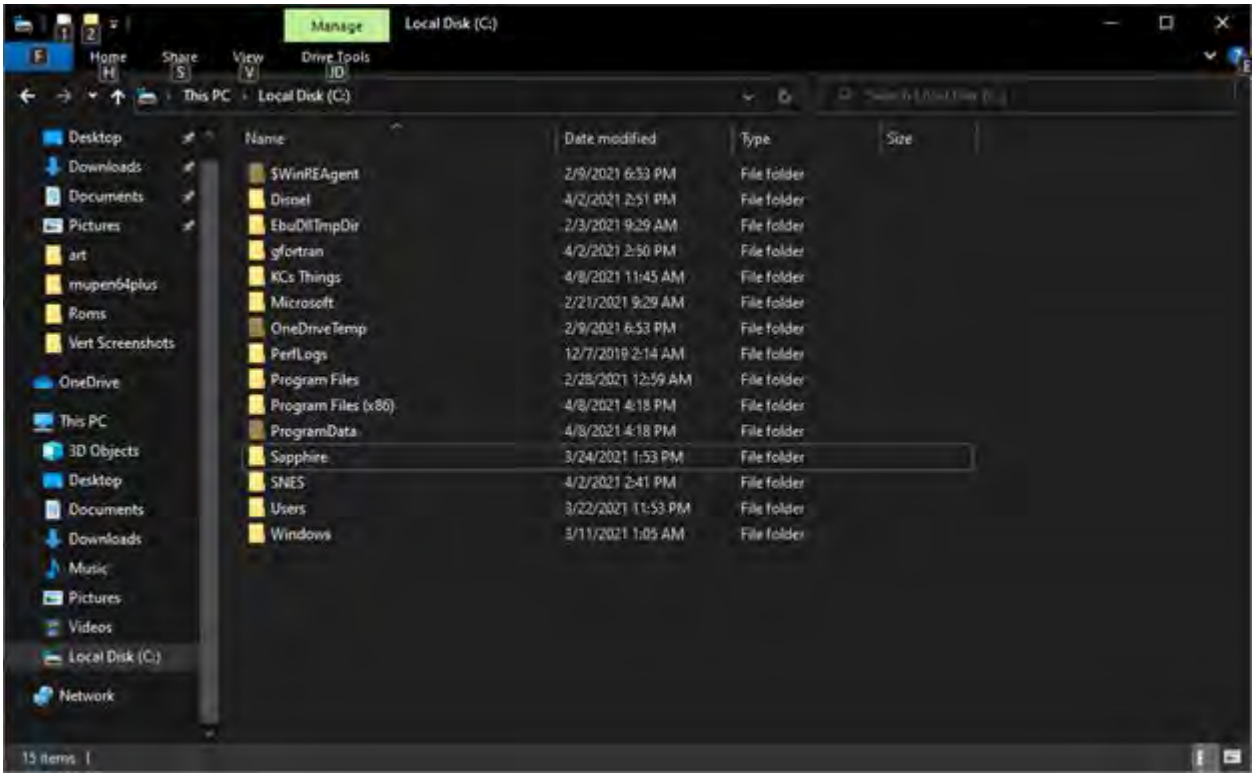
To get the current version of SAPHIRE 8, which is distributed through the SAPHIRE Users Group,

- Send a completed and signed non-disclosure agreement (NDA) to the U.S. Nuclear Regulatory Commission (USNRC) for approval prior to receiving an authorization to join the SAPHIRE Users Group.
- Locate a copy of the SAPHIRE NDA form on the Obtaining the Code page at the USNRC's public website.

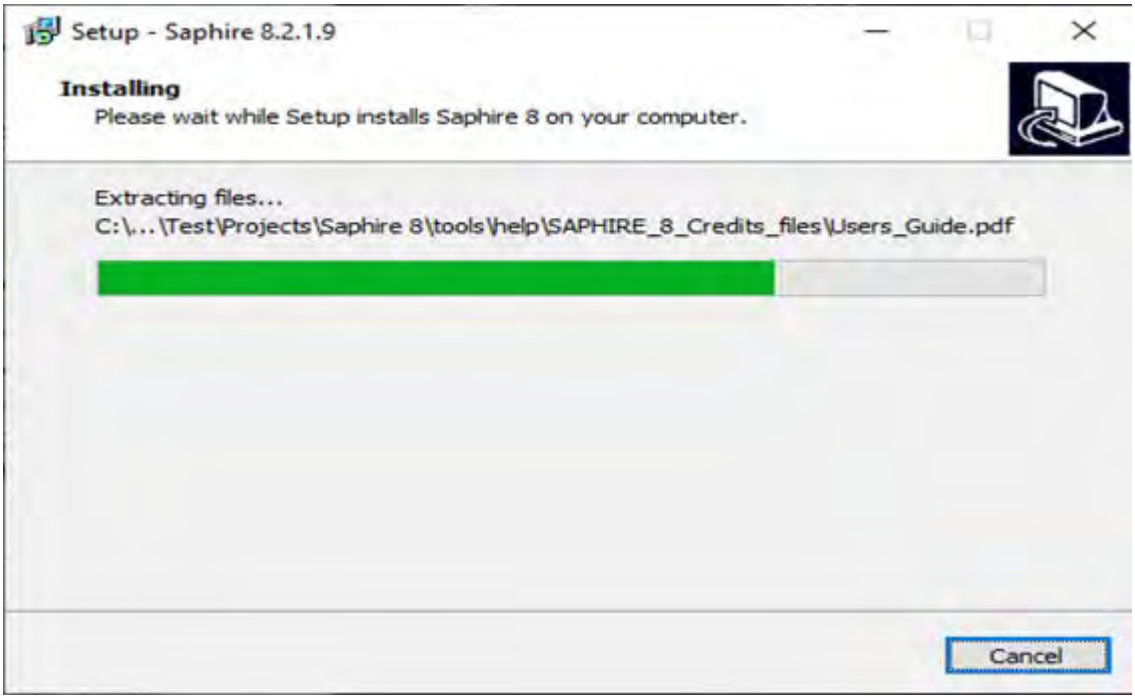
Once an NDA is on file with the USNRC, INL will proceed with establishing the SAPHIRE Users Group membership. Members will be granted access to the SAPHIRE Users Group website where they can download the current executable for SAPHIRE 8 and receive customer support. Customer support and access to updated SAPHIRE releases will continue throughout the membership period.

For U.S. Government departments and agencies, the U.S. NRC may grant exceptions to the requirement for SAPHIRE Users Group membership. These requests still require a completed and signed NDA form.

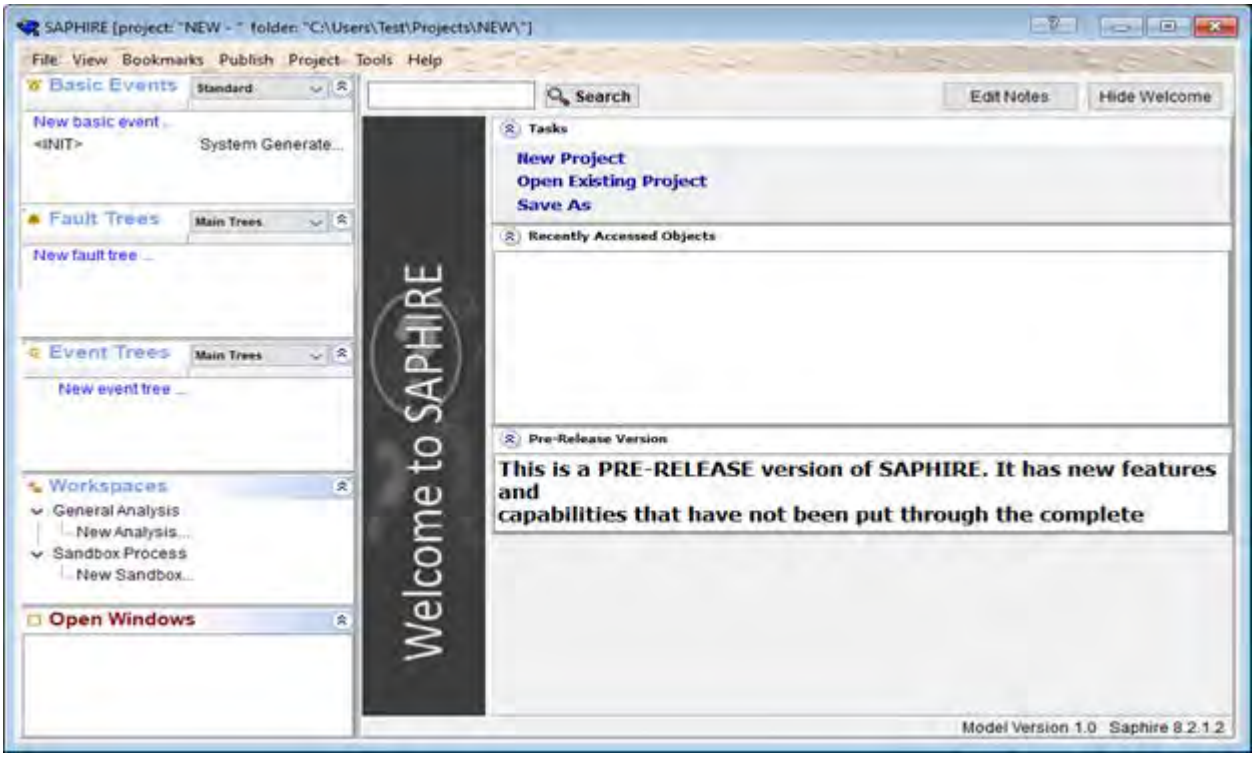
Once you have secured a copy of SAPHIRE, install it directly to the C: drive or the root drive. [5]



Installation screen for verification.



Open SAPHIRE for verification.

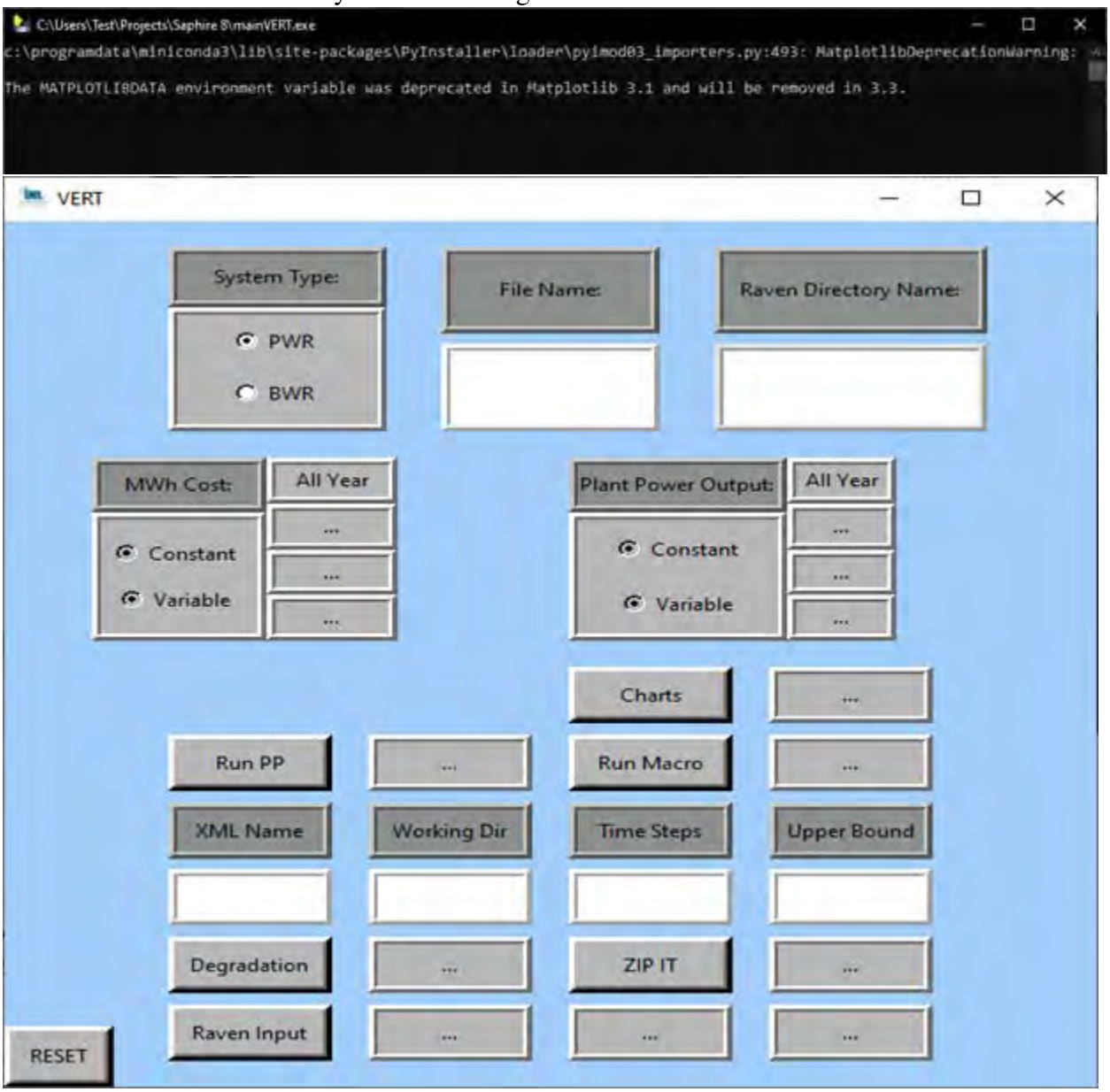


At this point, the installation of SAPHIRE is complete, and the program can be closed.

3.4 VERT

To obtain a copy of VERT, please contact Jaden Miller (milljad2@isu.edu).

Run the mainVERT.exe file by double clicking it.

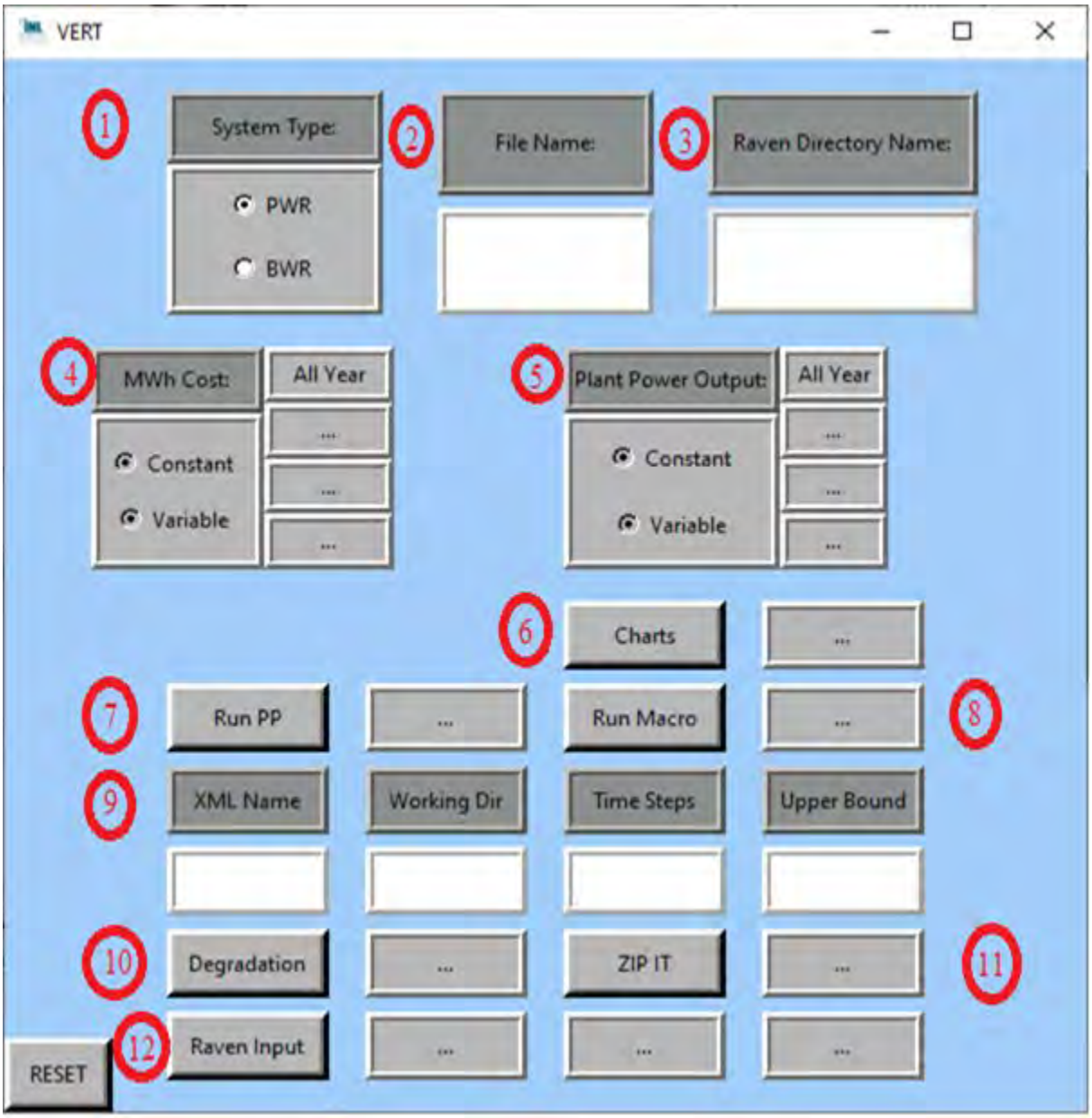


The VERT installation is now complete.

4 Using VERT

4.1 VERT Basics

The VERT window contains all the necessary steps to run VERT. The figure below has a number next to each of the VERT functions to run the user through what they do. It is important that the user does so in the order indicated, as the functions will build off each other.



1. **System Type** - The selection of the type of system VERT will be using; left click either PWR or BWR.

2. **File Name** - The file name of the SAPHIRE project.

3. **RAVEN Directory Name** - The name of the project folder in the RAVEN directory that contains the VERT.exe and the data.csv files.

4. **MWh Cost** - The user can select whether the cost per megawatt hour is fixed or variable throughout the year. If fixed, a single-entry box will appear. If variable, four boxes will open, allowing the user to specify a quarterly price.

5. **Power Plant Output** - The user can select from a constant or variable power output per year. If fixed, a single-entry box will appear. If variable, four boxed will open, allowing the user to specify a quarterly price.

6. **Charts** - This will generate the initial risk folder where the risk of the project has been quantified according to the risk formula in MWh/yr and the cause codes. If successful, the box to the right will say ”Complete,” and a new folder called ”initial risk” will be generated in the RAVEN project folder.

7. **Run PP** - This will generate a custom fault tree postprocessing script that will populate in a new window. If successful, the box to the right will say ”Complete.”

8. **Run Macro** - This will create the Macro file and place it into the correct directory. If successful, the box to the right will say ”Complete.”

9. Includes the following sections:

    a. **XML Name** - This will be the file name for the XML file that is generated.

    b. **Working Dir** - This will be the folder name where the run and output information are stored.

    c. **Time Steps** - The number of times RAVEN will iterate through SAPHIRE, typical numbers range from 20 - 25.

    d. **Upper Bound** - The number of years to be analyzed.

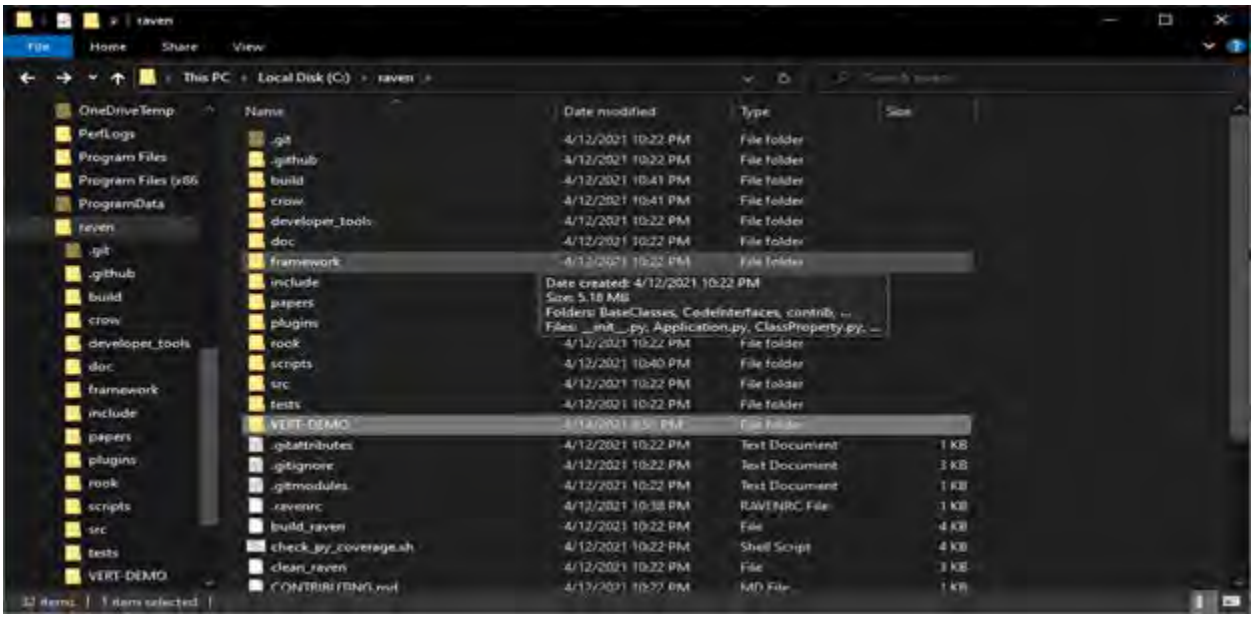
10. **Degradation** - This generates the degradation equations associated with the project and places them in the proper directory. This will create the ”Dump” folder and place it into the RAVEN project folder. If successful, the box to the right will say, ”Complete.”

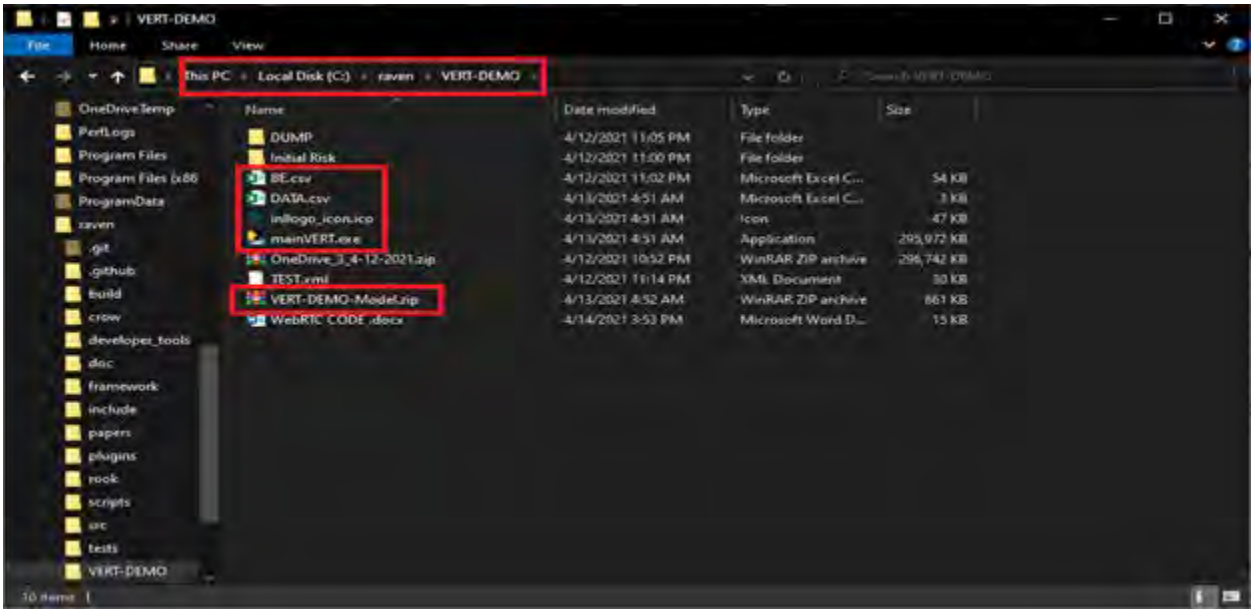
11. **ZIP IT** - This will zip the SAPHIRE project into a compressed folder placed into the SAPHIRE project folder. If successful, the box to the right will say ”Complete.”

12. **Raven Input** - This will create the .xml file that will be run in RAVEN. If successful, the box to the right will say ”zip found,” the box to the right of that will say ”macro found,” and the last box will say ”created file.”

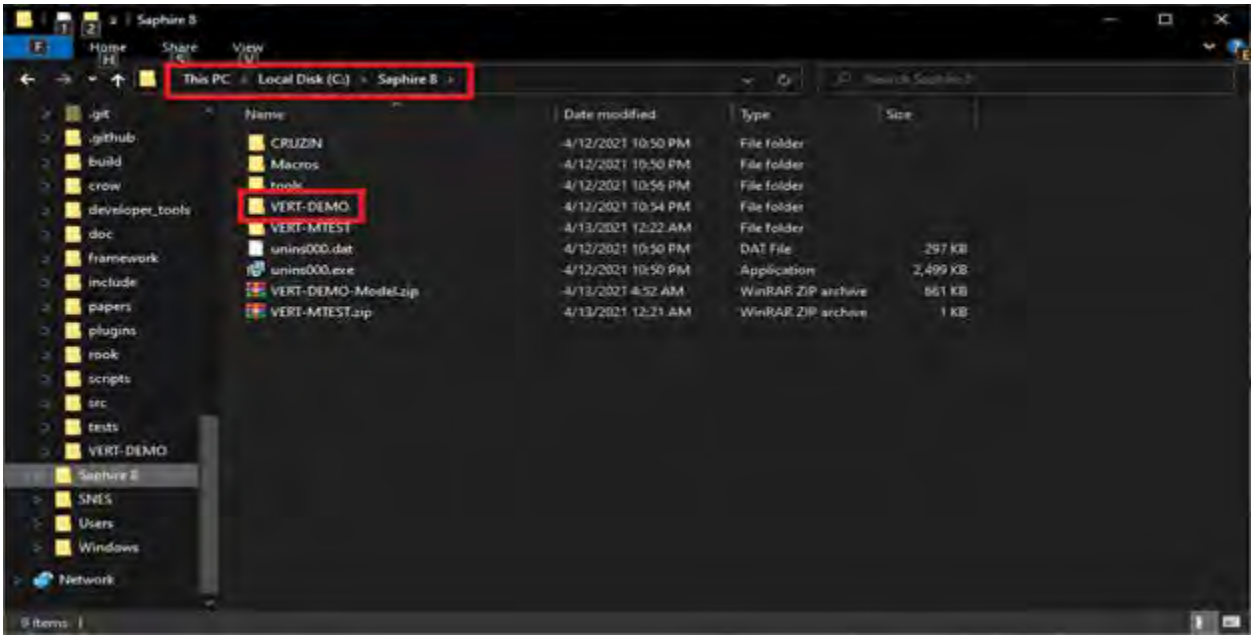
4.2 VERT Demonstration

This demonstration will ensure that VERT is properly installed and running correctly. The test will use the VERT-DEMO file to ensure a short computational time. The demo file needs to be extracted to the raven directory. This demo file can be received from Jaden Miller (milljad2@isu.edu) [6].

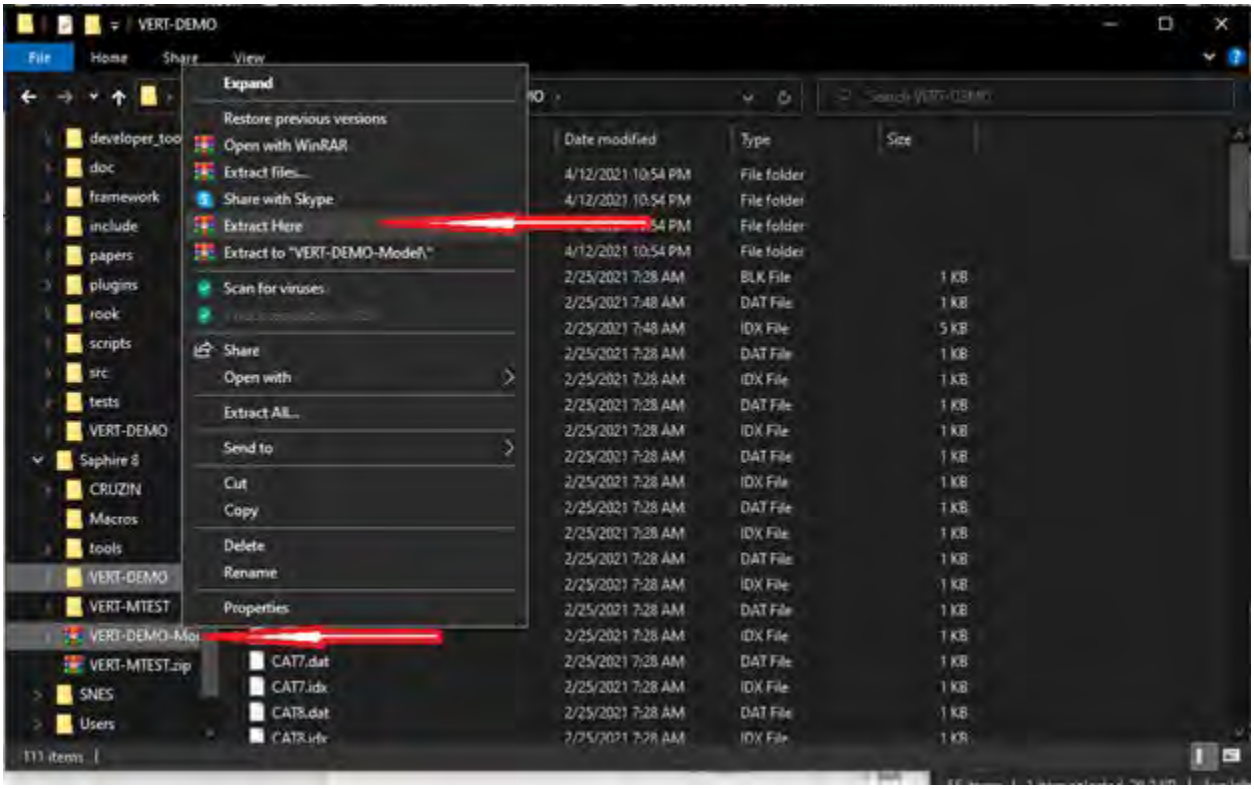




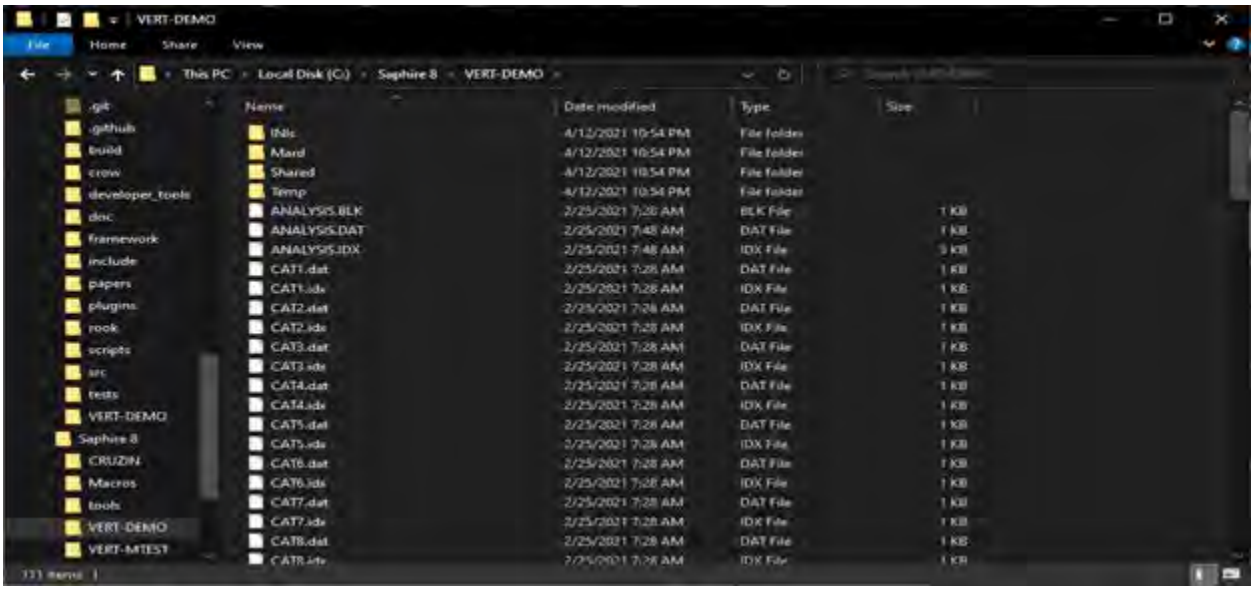
The file VERT-DEMO-Model.zip is the SAPHIRE demo model. The zip file needs to be relocated to the SAPHIRE directory and extracted there.



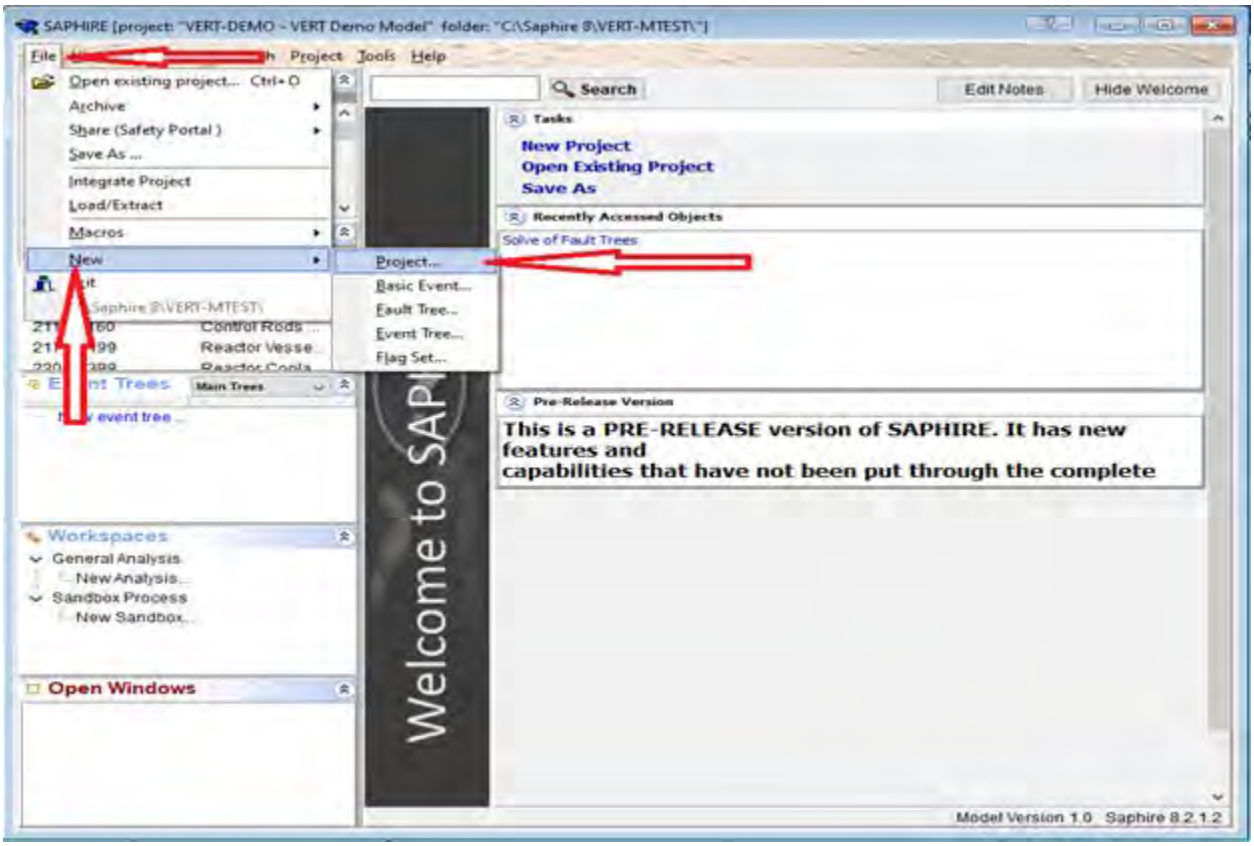
Right click on the VERT-DEMO-Model.zip file and select ”Extract Here.” This will create the VERT-DEMO folder.



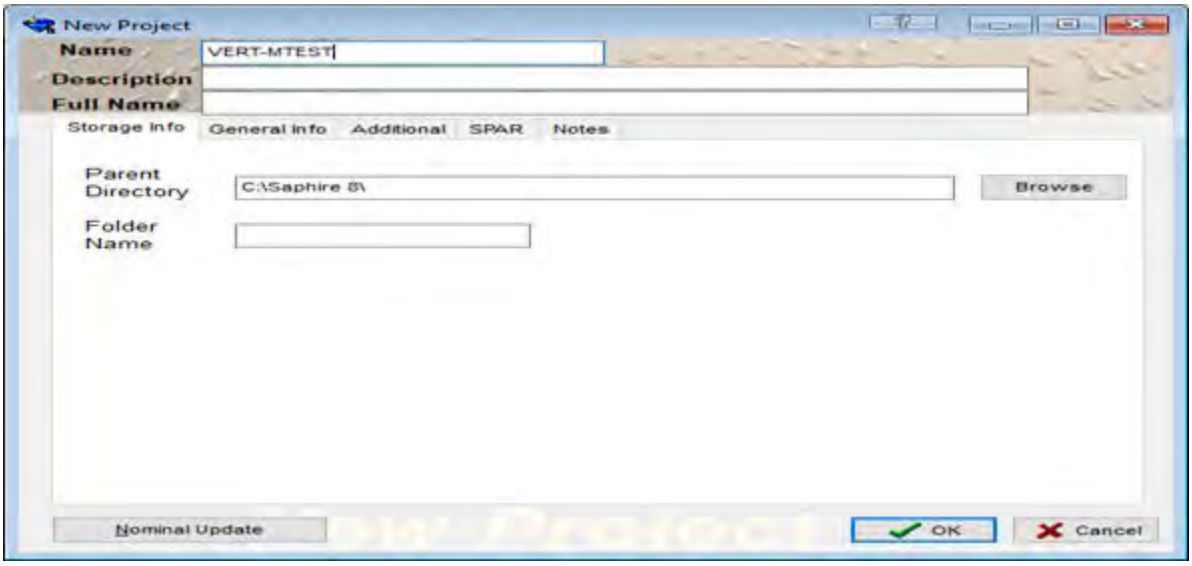
Once extracted, it should look like this.



The test model samples a single componont from each sub-system to reduce computational time requirements. To open the model, create a new SAPHIRE project. If not prompted on opening SAPHIRE, left click the ”File” tab and select ”New” and then ”Project.” [6]

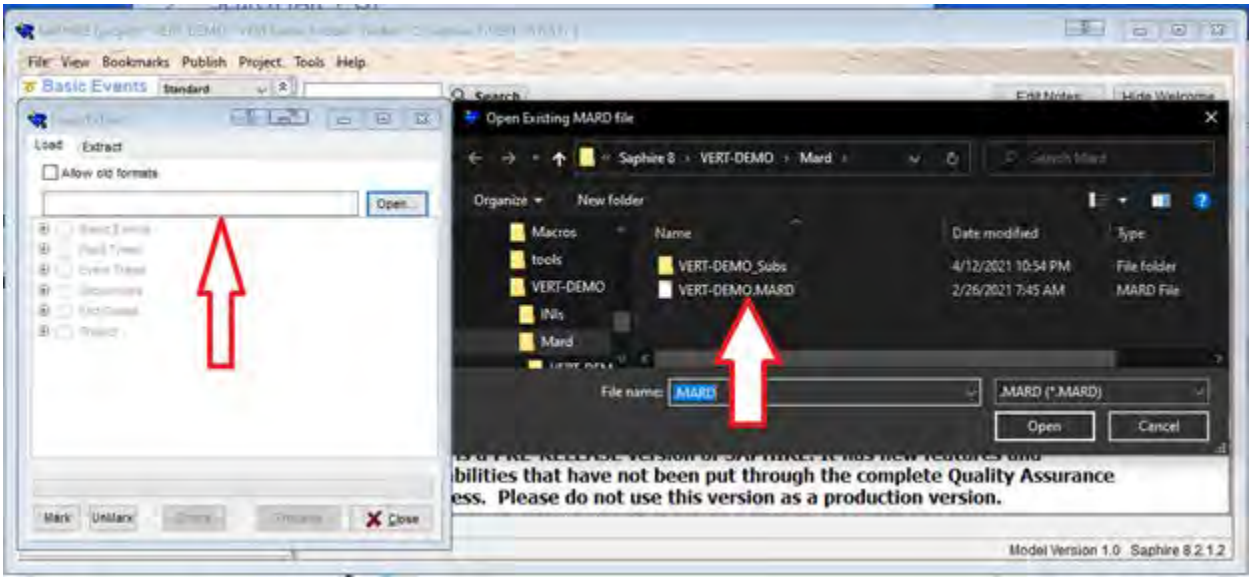


The project can be named anything. For this example, it is named VERT-MTEST.

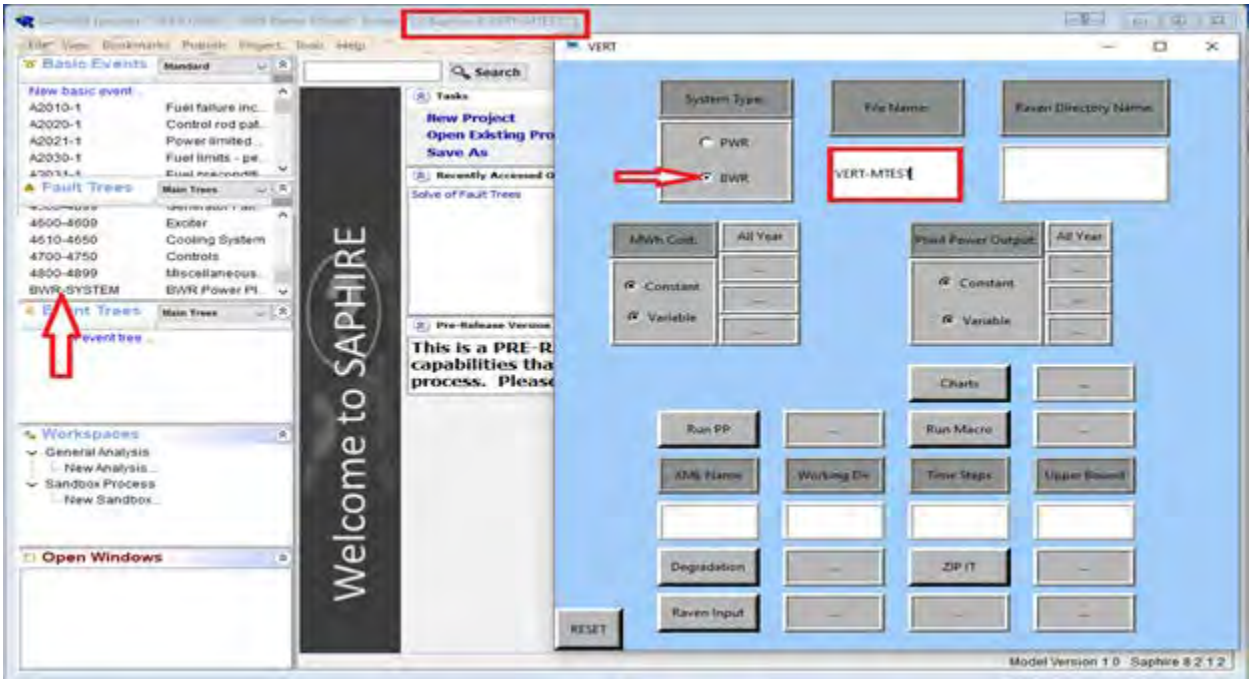


To load the project, left click on ”File” then ”Load/Extract.” Click on ”Open” and go to the ”Mard”

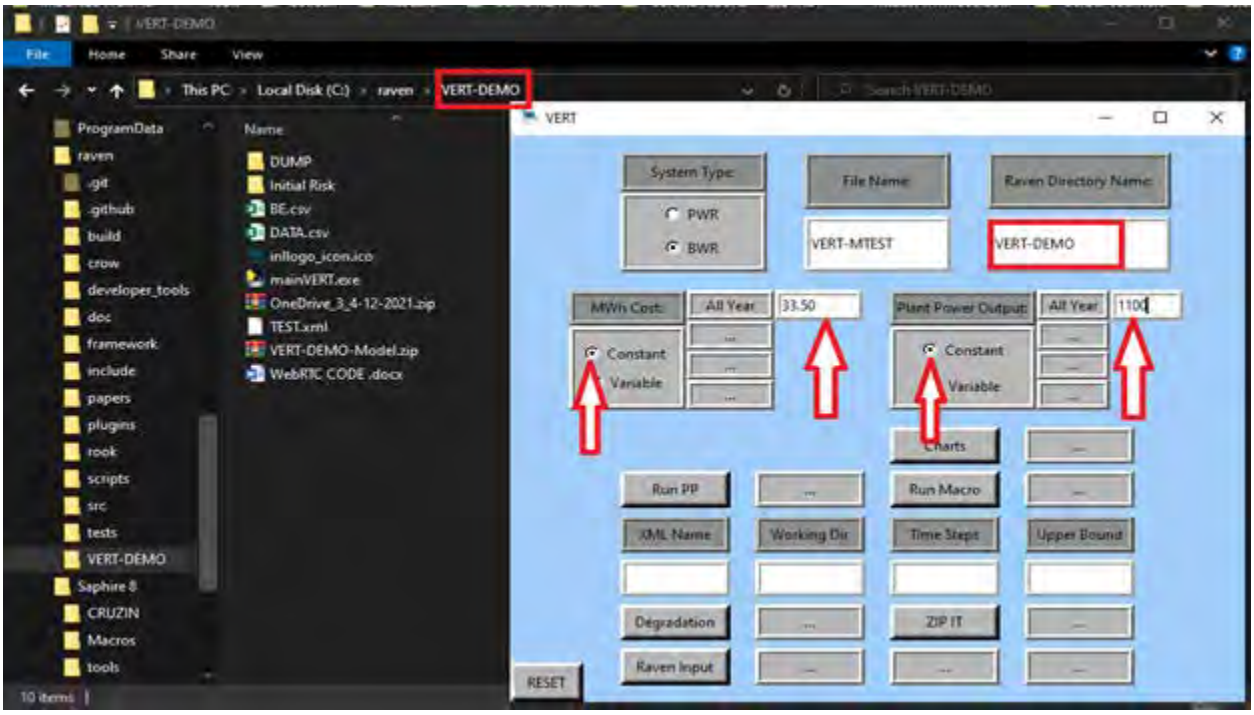
folder under VERT-DEMO and select VERT-DEMO.MARD.



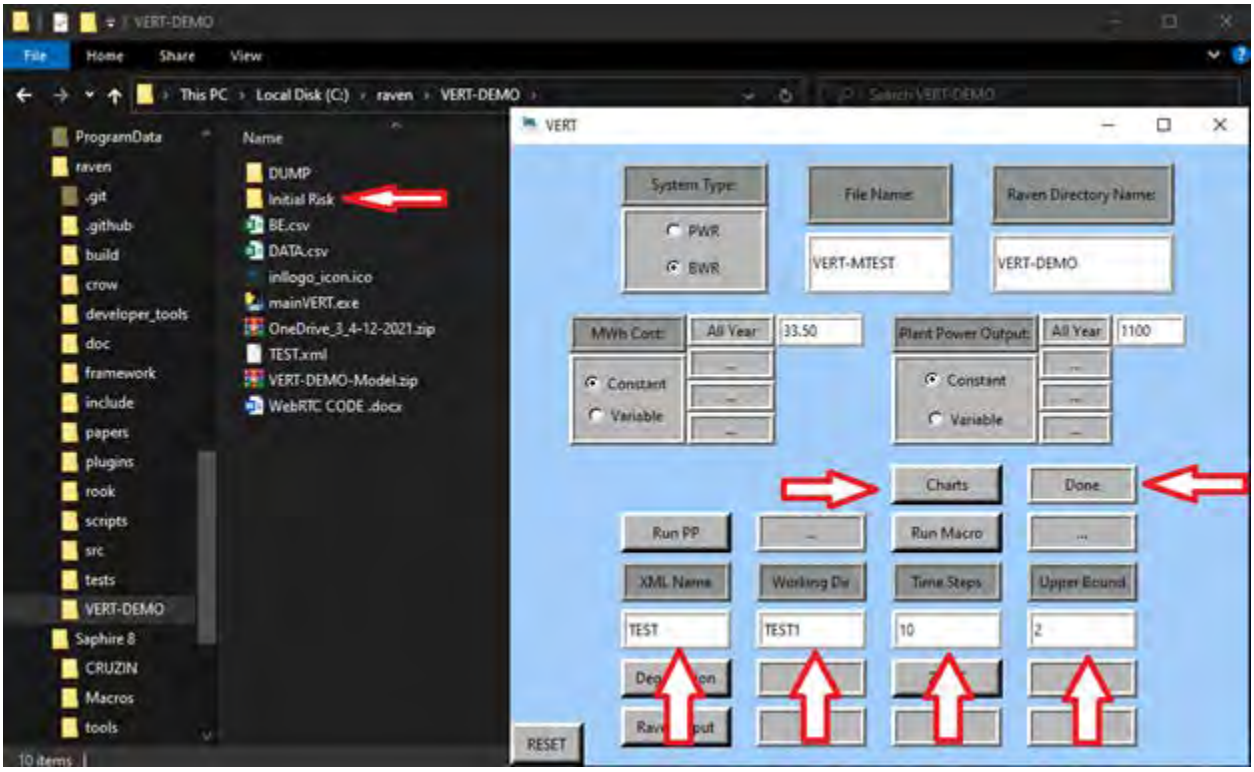
Launch VERT and select BWR. Then, input the file name, which will be the project name chosen in SAPHIRE.



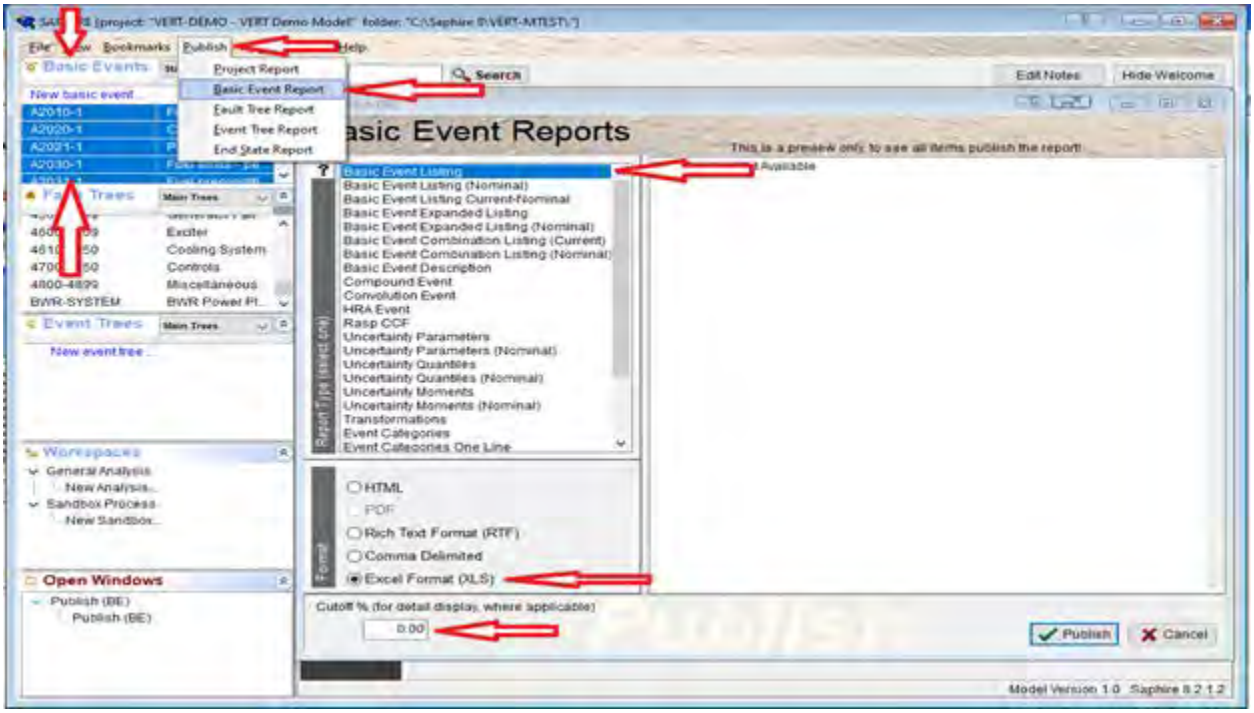
Fill out the ”Raven Directory Name” box with ”VERT-DEMO.” Select a constant MWh cost, and enter in 33.50 for a constant cost of $33.50 per megawatt hour. Then, select the constant power output, and enter 1100 for a constant power output of 1,100 MWe.



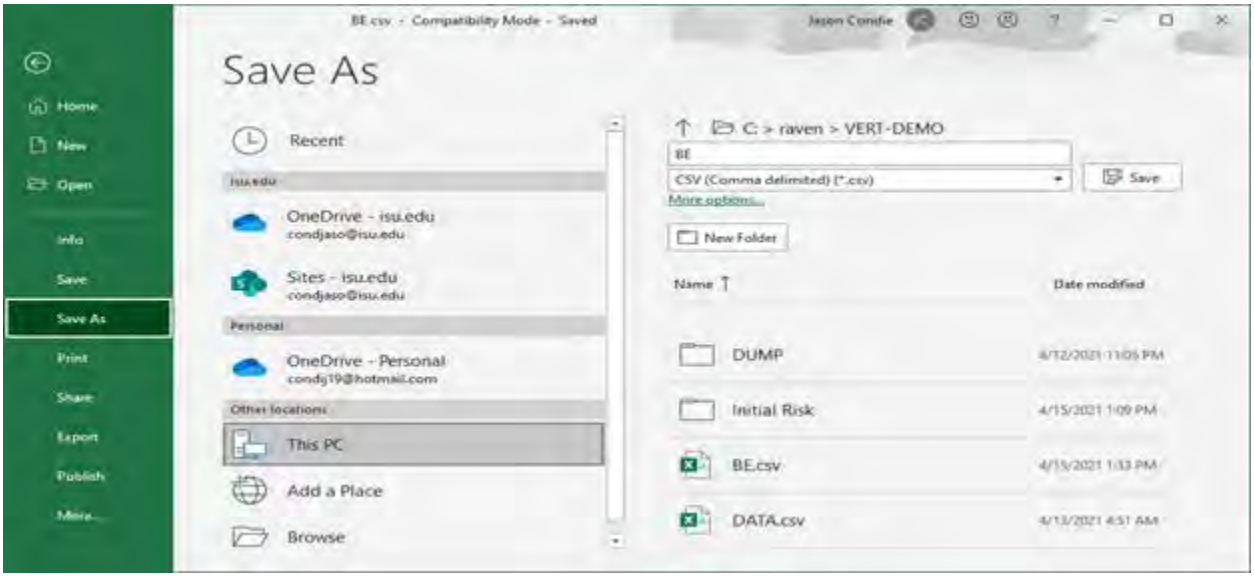
Fill in ”XML Name” and ”Working Dir” (named TEST and TEST1, respectively, in this example). Fill in ”Time Steps”, using 10 for this example, and then enter 2 in the ”Upper Bound” box. Click on ”Charts” to generate the initial risk folder.



We now need to create a basic event report. In SAPHIRE, click on a basic event in the basic event window and then press Ctrl + A. This will highlight all the basic events in our boiling-water reactor model. Left click on ”Publish,” and select ”Basic Event Report.” Select ”Basic Event Listing,” ”Excel Format,” and 0% cutoff. Then, click on ”Publish.”

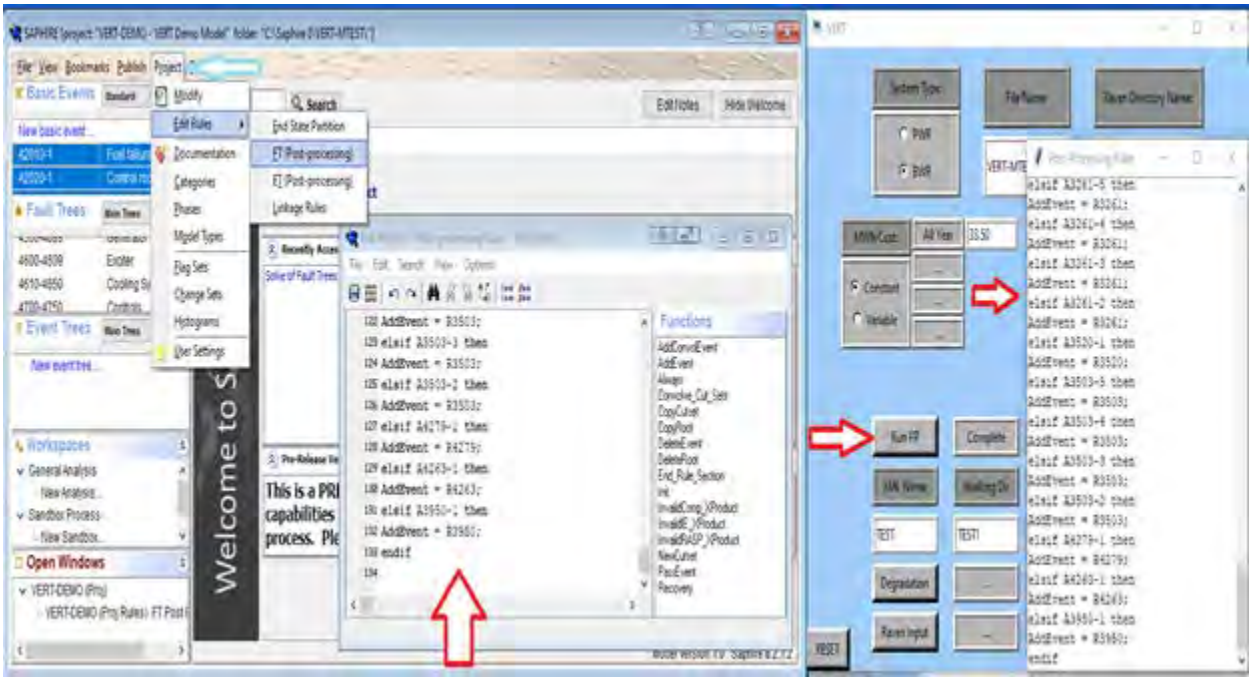


Save this Excel file to the VERT-DEMO folder in the RAVEN directory. It will need to be saves as a CSV file and not an XLS file. The name will also have to be changed to ”BE” (the full filename would therefore be ”BE.csv.” To do this, click on ”File” in the Excel sheet then ”Save As.”

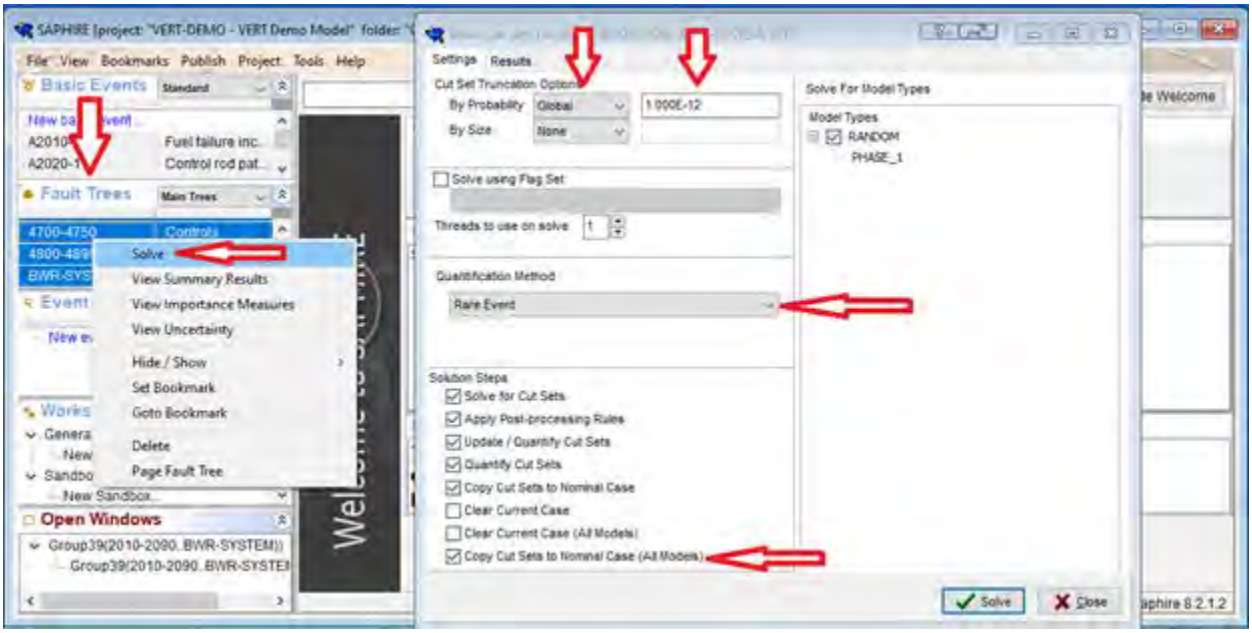


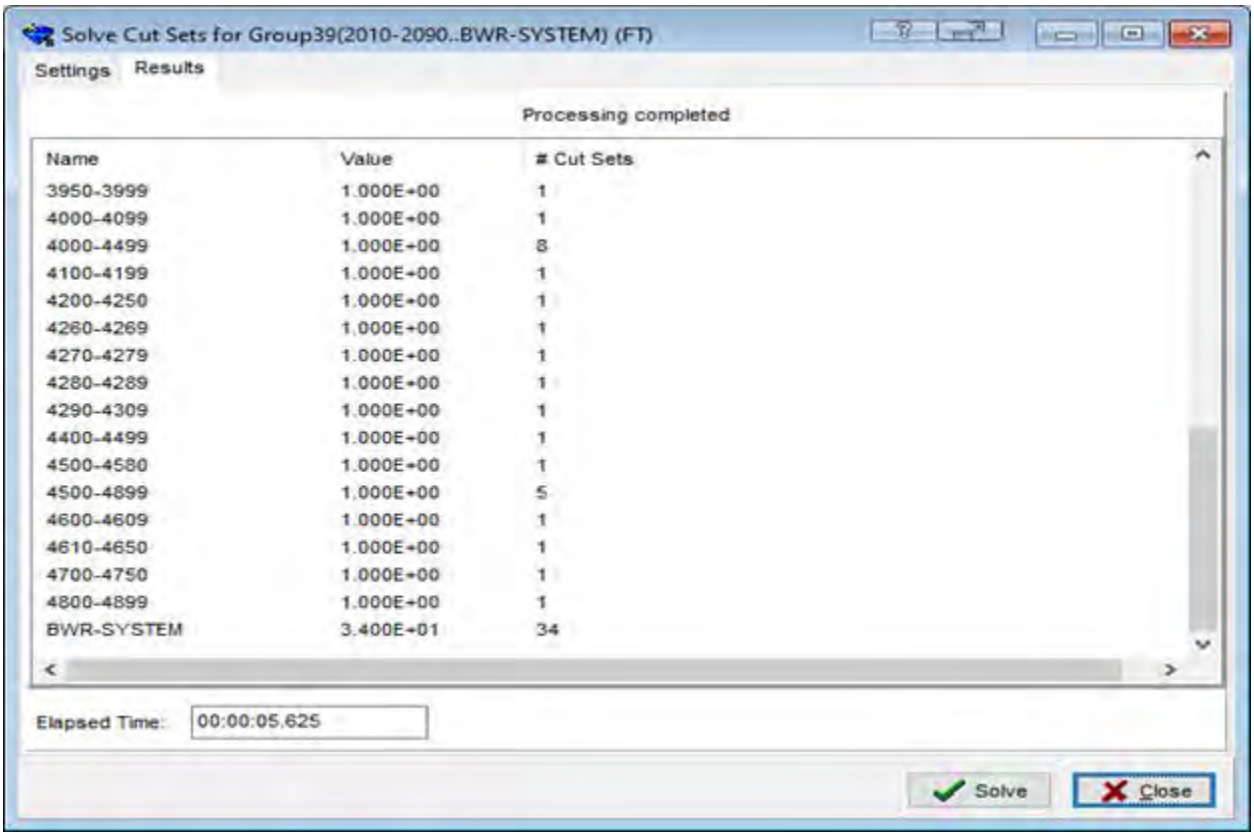
The next step imposes the VERT ruleset onto the SAPHIRE postprocessing rules. Click the ”Run PP” button in VERT to generate the postprocessing rules. In SAPHIRE, left click on ”Project” then ”FT(Post-processing)” to bring up the SAPHIRE postprocessing window. Copy and place the

VERT post-processing script into the SAPHIRE post-processing window. Then, left click ”File” in SAPHIRE and ”Save.”

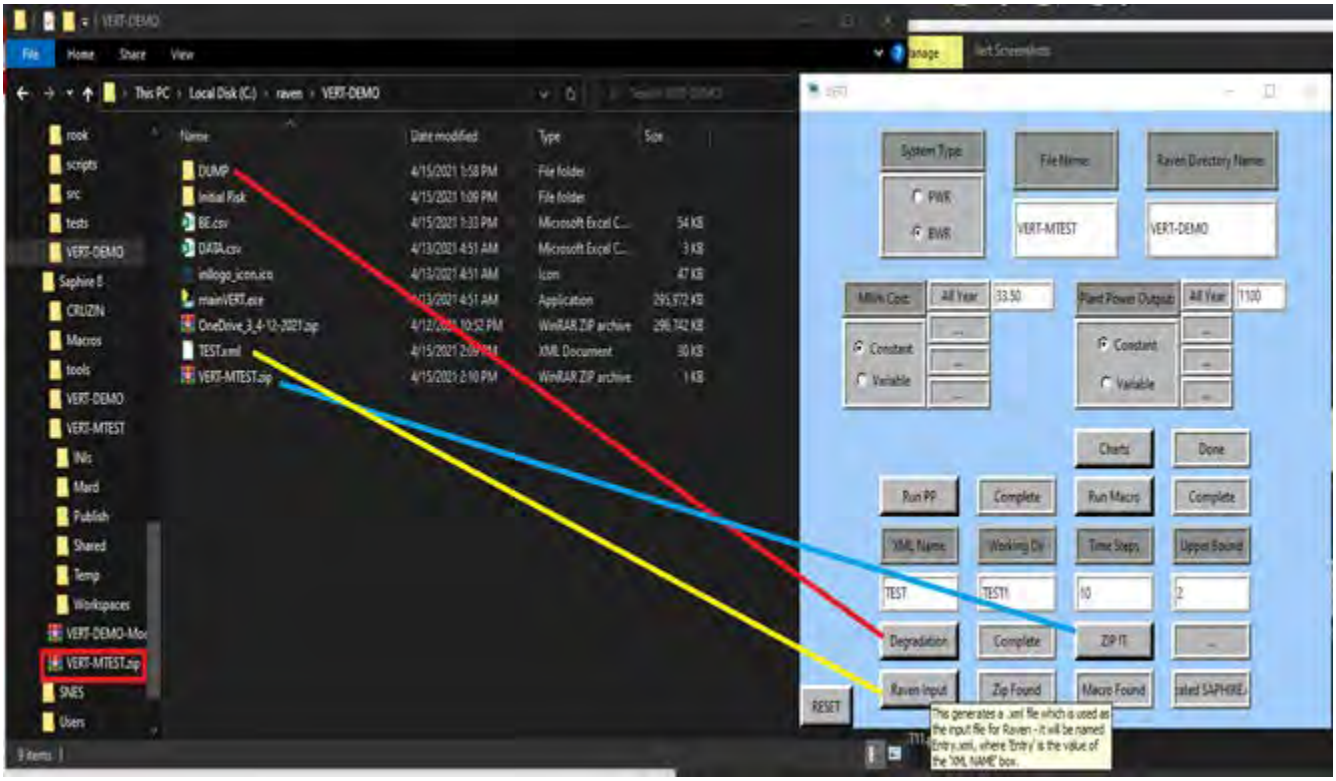


Now we must solve the SAPHIRE model by highlighting one of the fault trees and pressing Ctrl + A to select them all. Right click on the fault tree and select ”Solve.” Choose a Global Probability of 1.000E-12. Set the quantifiation method to ”Rare Event” and check the ”Copy Cut Sets to Nominal Case (All Models).” Then, click ”Solve.”



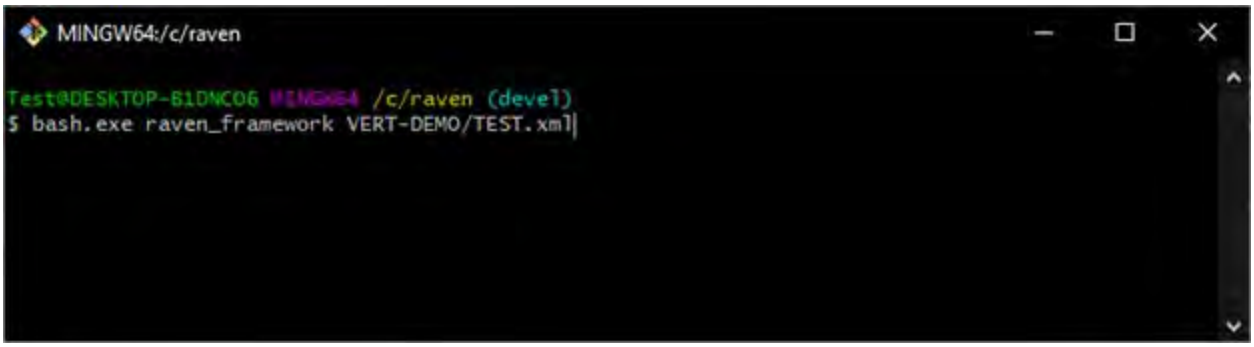


SAPHIRE must now be closed. In VERT, click on ”Run Macro” and then ”Degradation” to generate the ”DUMP” folder. Then, click the ”ZIP IT” button to create the .zip file. This must be placed in the SAPHIRE directory. Finally, click the ”Raven Input” button to generate the .xml file. The ”Raven Input” can take a few minutes to process.

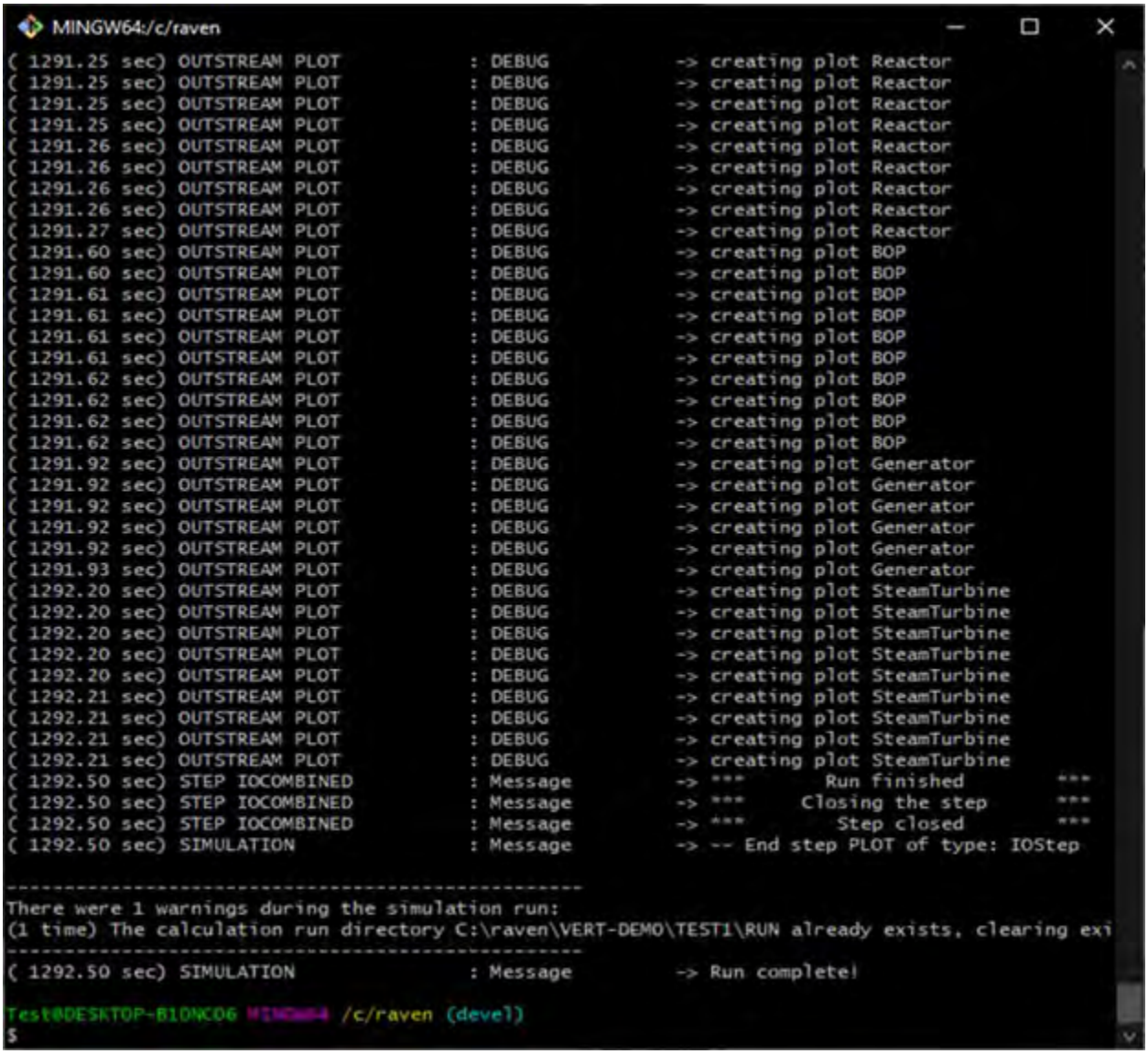


Now we must input the .xml file in RAVEN to run the analysis. Open RAVEN by double clicking on the ”raven_framework.bat” file and type in the name of the .xml file.

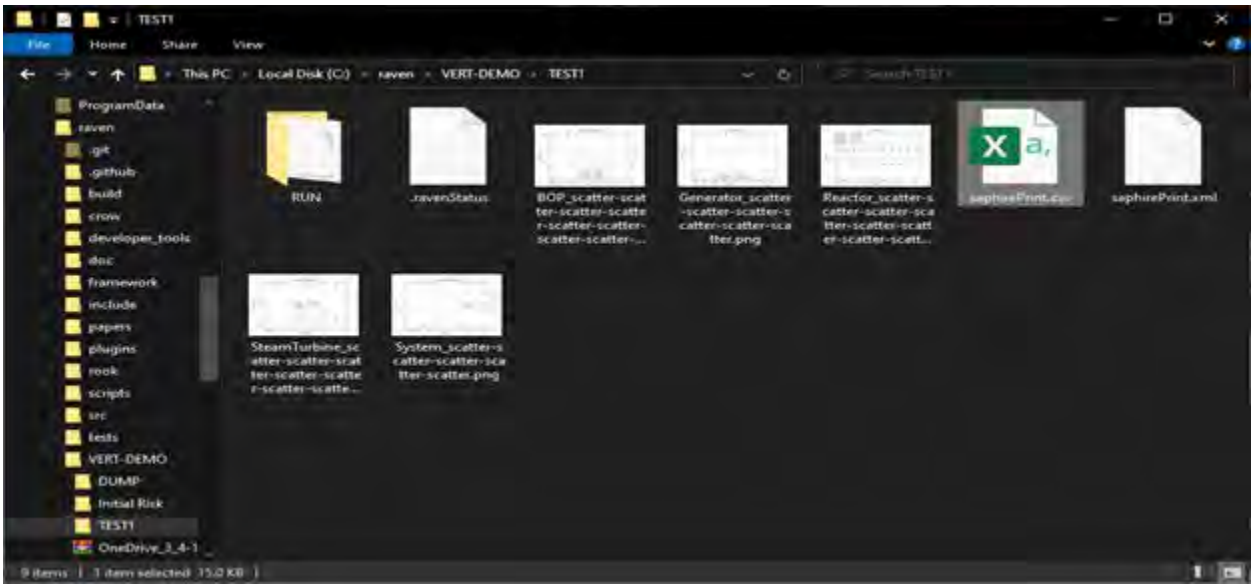
This can be done in the GitBash command window by entering the command ”bash.exe raven_framework <xml filename here>.”

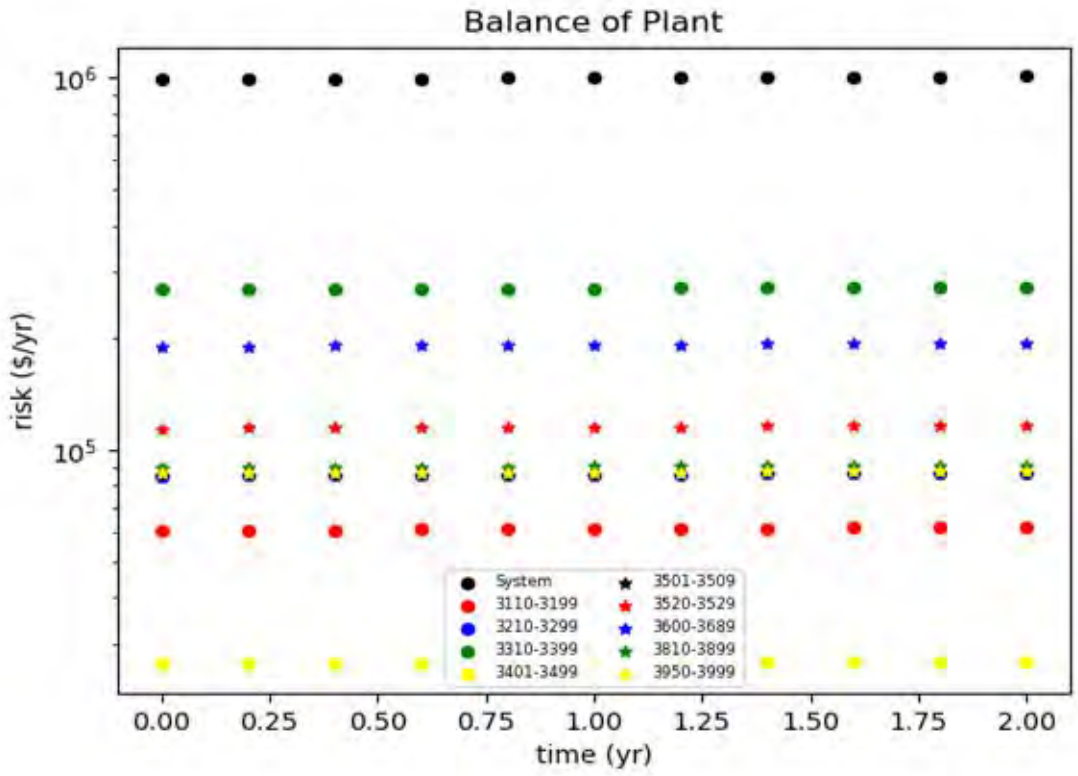


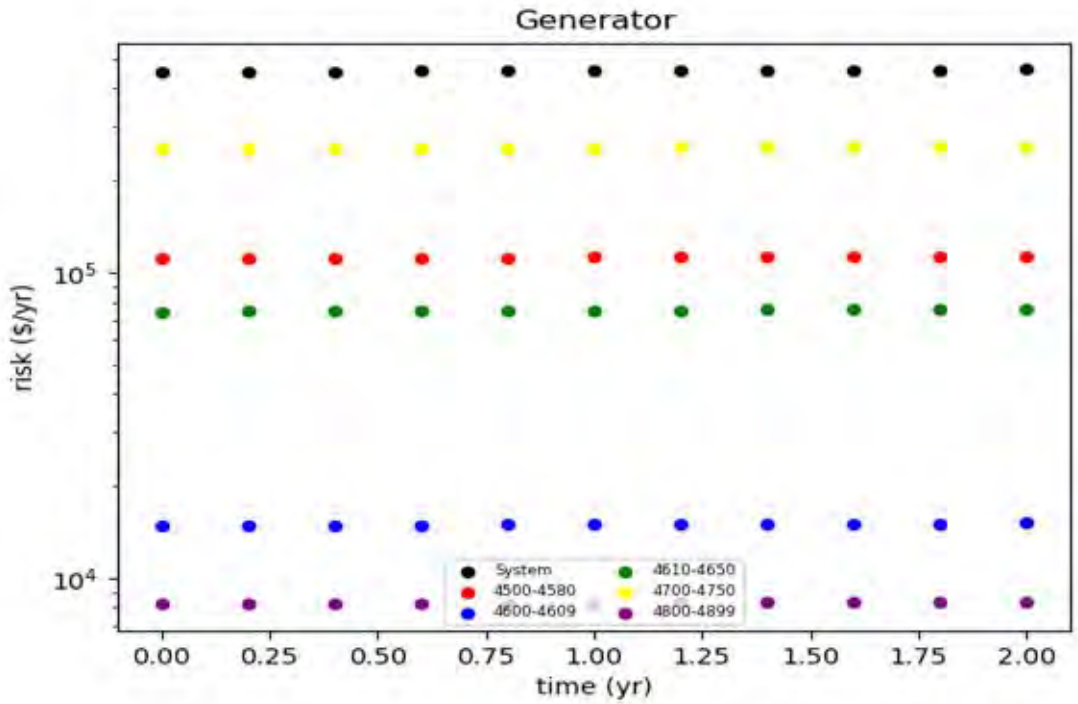
Completion screen for verification.

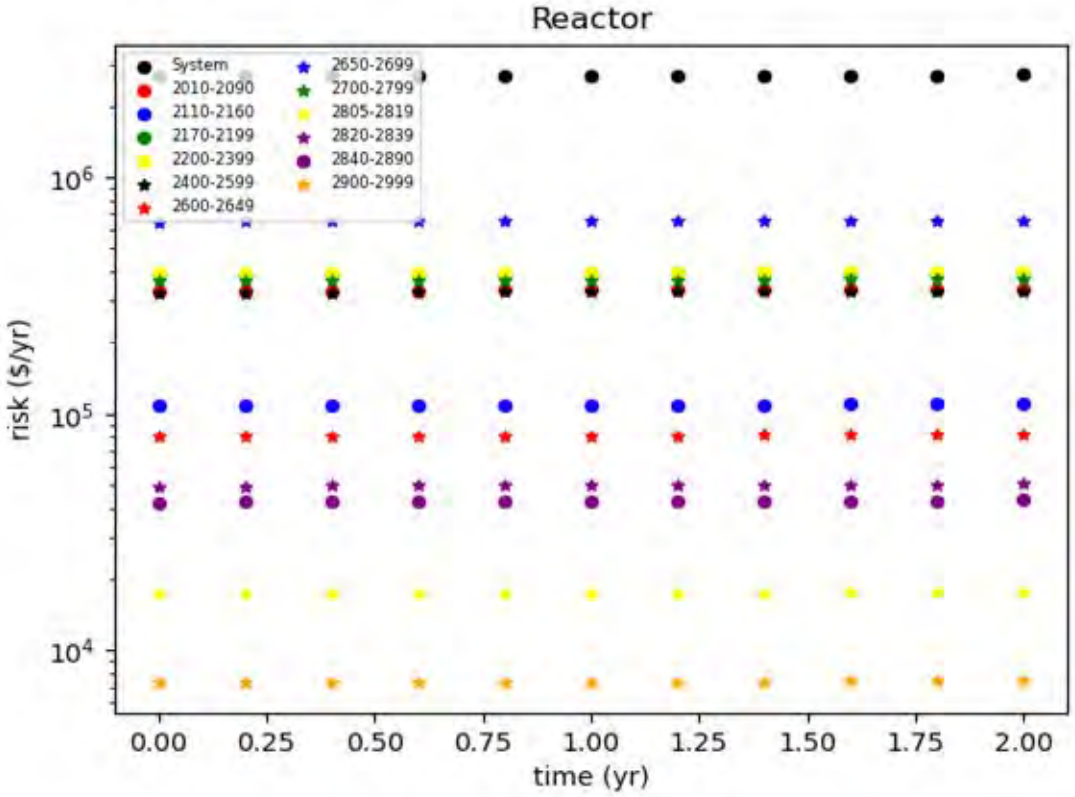


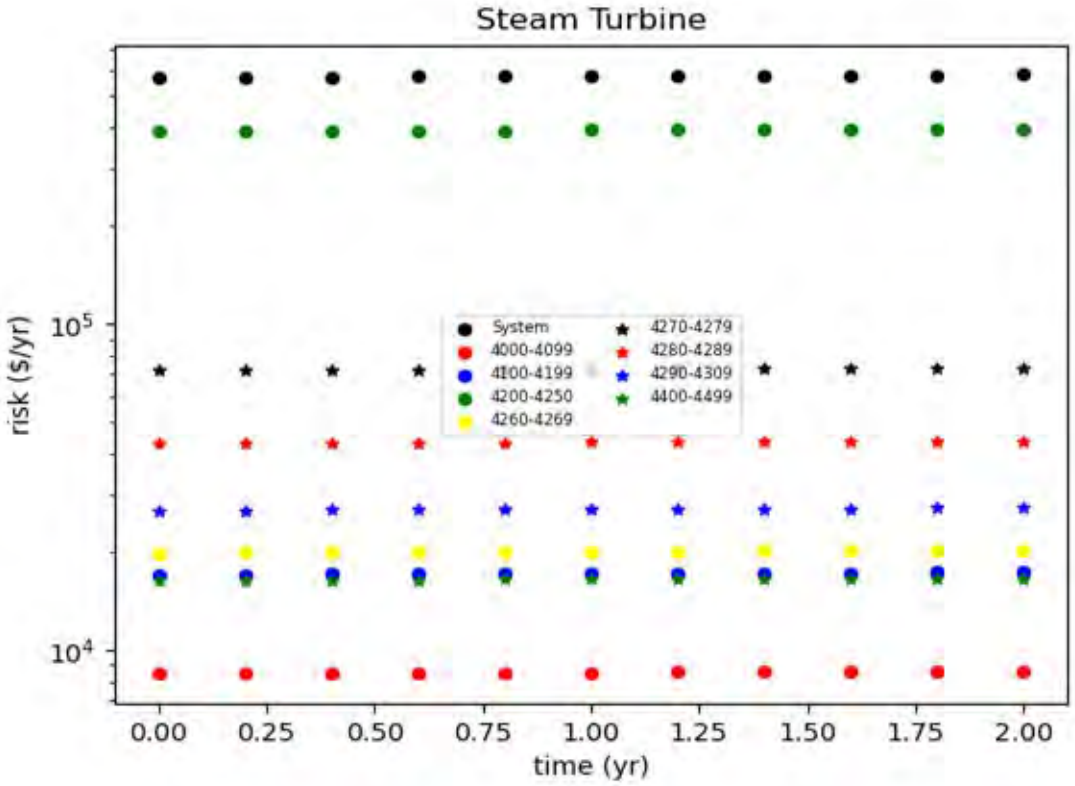
This will generate a series of files for the runs, graphs, and an Excel sheet that shows the probabilities of events with a cost that is placed within the RAVEN directory. Check your results against those here to ensure proper working order of VERT.

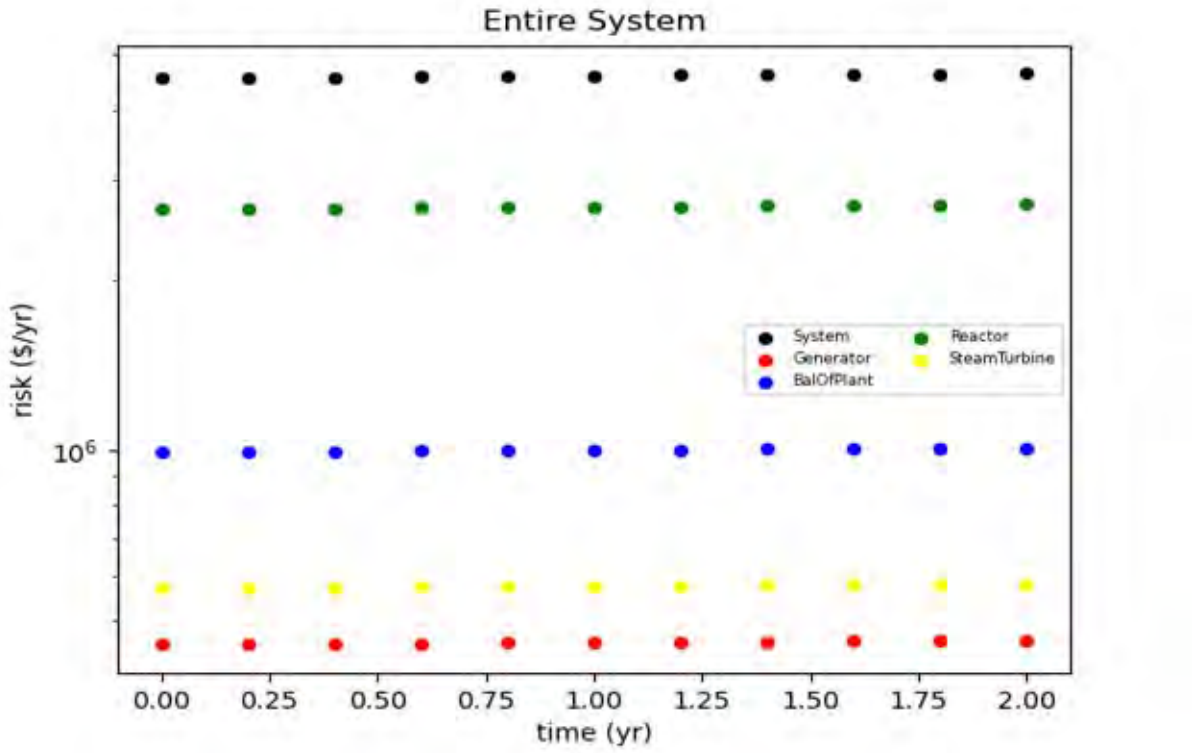





Generator
risk ($/yr)
10^5
10^4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)
System
4500-4580
4600-4609
4610-4650
4700-4750
4800-4899


Reactor
System
2010-2090
2110-2160
2170-2199
2200-2399
2400-2599
2600-2649
2650-2699
2700-2799
2805-2819
2820-2839
2840-2890
2900-2999
risk ($/yr)
10^6
10^5
10^4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)


Steam Turbine
risk ($/yr)
10^5
10^4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)
System
4000-4099
4100-4199
4200-4250
4260-4269
4270-4279
4280-4289
4290-4309
4400-4499



t	2070	R20 70	2120	R21 20	2171	R217 1
0	0.03 9521	844 4586	0.02 2448	463 4048	0.00 7093	5134 4574
0.2	0.03 9599	844 4586	0.02 2493	463 4048	0.00 7107	5134 4574
0.4	0.03 9677	844 4586	0.02 2537	463 4048	0.00 7121	5134 4574
0.6	0.03 9754	844 4586	0.02 2582	463 4048	0.00 7135	5134 4574
0.8	0.03 9832	844 4586	0.02 2626	463 4048	0.00 7149	5134 4574
1	0.03 9911	844 4586	0.02 2671	463 4048	0.00 7164	5134 4574
1.2	0.03 9989	844 4586	0.02 2716	463 4048	0.00 7178	5134 4574
1.4	0.04 0067	844 4586	0.02 2761	463 4048	0.00 7192	5134 4574
1.6	0.04 0146	844 4586	0.02 2806	463 4048	0.00 7207	5134 4574
1.8	0.04 0225	844 4586	0.02 2851	463 4048	0.00 7221	5134 4574
2	0.04 0304	844 4586	0.02 2896	463 4048	0.00 7235	5134 4574

2	R	2	R	2	R	2	R
230	2230	411	2411	629	2629	650	2650
0.	14	0.	37	0.	47	0.	12
025784	750373	008137	676385	001577	906289	053148	838611
0.	14	0.	37	0.	47	0.	12
025835	750373	008153	676385	00158	906289	053251	838611
0.	14	0.	37	0.	47	0.	12
025886	750373	008169	676385	001583	906289	053355	838611
0.	14	0.	37	0.	47	0.	12
025938	750373	008185	676385	001586	906289	053459	838611
0.	14	0.	37	0.	47	0.	12
025989	750373	008202	676385	001589	906289	053563	838611
0.	14	0.	37	0.	47	0.	12
02604	750373	008218	676385	001592	906289	053667	838611
0.	14	0.	37	0.	47	0.	12
026092	750373	008234	676385	001596	906289	053772	838611
0.	14	0.	37	0.	47	0.	12
026143	750373	008251	676385	001599	906289	053876	838611
0.	14	0.	37	0.	47	0.	12
026195	750373	008267	676385	001602	906289	053981	838611
0.	14	0.	37	0.	47	0.	12
026247	750373	008284	676385	001605	906289	054086	838611
0.	14	0.	37	0.	47	0.	12
026298	750373	0083	676385	001608	906289	054192	838611

27	R2	28	R	28	R2	28	R2
00	700	15	2815	33	833	43	843
0.	36	0.	4	0.	12	0.	21
009423	501482	003684	416714	003684	643382	001848	524470
0.	36	0.	4	0.	12	0.	21
009441	501482	003691	416714	003691	643382	001852	524470
0.	36	0.	4	0.	12	0.	21
00946	501482	003698	416714	003698	643382	001855	524470
0.	36	0.	4	0.	12	0.	21
009479	501482	003706	416714	003706	643382	001859	524470
0.	36	0.	4	0.	12	0.	21
009498	501482	003713	416714	003713	643382	001863	524470
0.	36	0.	4	0.	12	0.	21
009517	501482	003721	416714	003721	643382	001866	524470
0.	36	0.	4	0.	12	0.	21
009536	501482	003728	416714	003728	643382	00187	524470
0.	36	0.	4	0.	12	0.	21
009555	501482	003736	416714	003736	643382	001874	524470
0.	36	0.	4	0.	12	0.	21
009574	501482	003743	416714	003743	643382	001878	524470
0.	36	0.	4	0.	12	0.	21
009593	501482	00375	416714	00375	643382	001881	524470
0.	36	0.	4	0.	12	0.	21
009612	501482	003758	416714	003758	643382	001885	524470

5 Modifying VERT

VERT uses the data collected from the GADS program to build the initial risk models and cause codes. These can be found in the DATA.csv file that is included in the generic projects and the demo. This data can be edited through Excel for a tailored response to the project. [7]

Nuclear Reactor	
Description	**Cause Code Range**
Core/Fuel	2010-2090
Control Rods and Drives	2110-2160
Reactor Vessel and Internals	2170-2199
Reactor Coolant System	2200-2399
Steam Generators and Steam System	2400-2599
Core Cooling/Safety Injection	2600-2649
Electrical Safety Systems	2650-2699
Containment System	2700-2799
Chemical and Volume Control/Reactor Water Cleanup	2805-2819
Nuclear Cooling Water Systems	2820-2839
Auxiliary Systems	2840-2890
Miscellaneous (Reactor)	2900-2999

Balance of Plant	
Description	**Cause Code Range**
Condensing System	3110-3199
Circulating Water Systems	3210-3299
Condensate System	3310-3399
Feedwater System	3401-3499
Heater Drain Systems	3501-3509
Extraction Steam	3520-3529
Electrical	3600-3689
Auxiliary Systems	3810-3899
Miscellaneous (Balance of Plant)	3950-3999

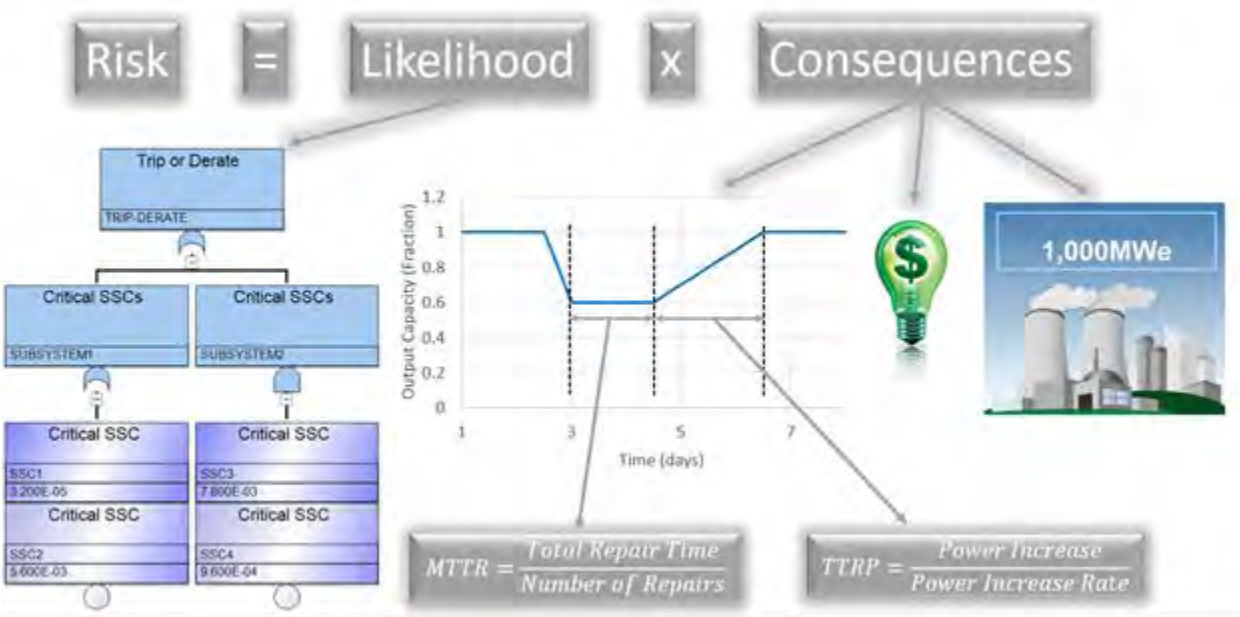
Generator	
Description	**Cause Code Range**
Generator	4500-4580
Exciter	4600-4609
Cooling System	4610-4650
Controls	4700-4750
Miscellaneous (Generator)	4800-4899

Steam Turbine	
Description	**Cause Code Range**
High Pressure Turbine	4000-4099
Intermediate Pressure Turbine	4100-4199
Low Pressure Turbine	4200-4250
Valves	4260-4269
Piping	4270-4279
Lube Oil	4280-4289
Controls	4290-4309
Miscellaneous (Steam Turbine)	4400-4499

The VERT code is stored within a GitHub repository and can be accessed with permissions from Jaden Miller (milljad2@isu.edu). The code is written in Python and all functions outside of SAPHIRE and RAVEN can be edited there. The base code contains the equations for the consequences and risks where:

- consequences = capacity(MWe)*Derate Fraction*(Mean time to repair + Time to restore power)*electricity costs [7]
- risk = likelihood*consequences [7]

The risk requires the input of SAPHIRE to generate the likelihood figure.



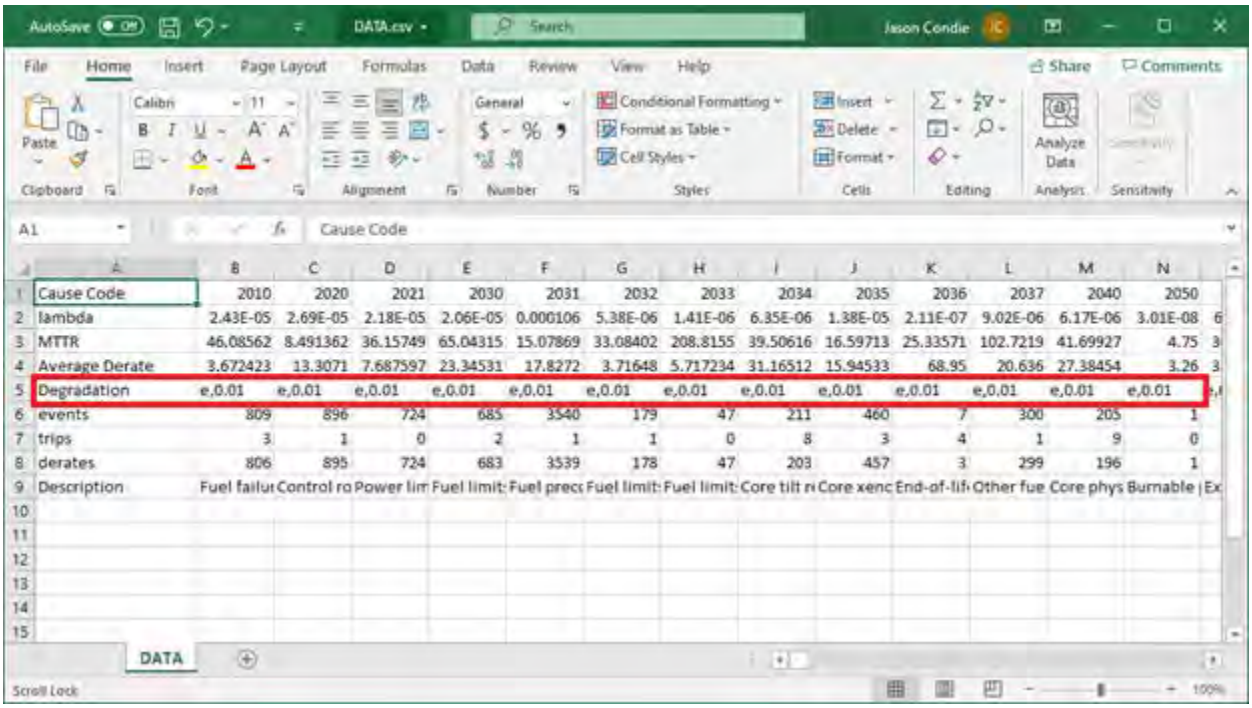
The base code also contains the functionality for the main VERT window. This includes the post-processing generation that cn be changed here for a change in generation of the script or in the

SAPHIRE post-processing editor for a single project. The basic events and fault trees must be edited within SAPHIRE. The base code contains a generator for the .xml file for RAVEN to run. Modifications to the code will result in a change to the generation of the .xml file or can be altered in the .xml file itself using a text editor to make a change for a single project. Current builds of VERT use a uniform distributions and a grid sampler.

VERT can integrate degradation into the component analysis. VERT is currently capable of supporting two different equation types: linear and exponential lambda degradation (failure rates). [7]

$$\lambda(t) = \lambda_0(1 + bt)$$
$$\lambda(t) = \lambda_0 e^{bt}$$

This can be found in the DATA.csv file shown in the figure below.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Cause Code	2010	2020	2021	2030	2031	2032	2033	2034	2035	2036	2037	2040	2050
2	lambda	2.43E-05	2.69E-05	2.18E-05	2.06E-05	0.000106	5.38E-06	1.41E-06	6.35E-06	1.38E-05	2.11E-07	9.02E-06	6.17E-06	3.01E-08
3	MTTR	46.08562	8.491362	36.15749	65.04315	15.07869	33.08402	208.8155	39.50616	16.59713	25.33571	102.7219	41.69927	4.75
4	Average Derate	3.672423	13.3071	7.687597	23.34531	17.8272	3.71648	5.717234	31.16512	15.94533	68.95	20.636	27.38454	3.26
5	Degradation	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01	e,0.01
6	events	809	896	724	685	3540	179	47	211	460	7	300	205	1
7	trips	3	1	0	2	1	1	0	8	3	4	1	9	0
8	derates	806	895	724	683	3539	178	47	203	457	3	299	196	1
9	Description	Fuel failu	Control ro	Power lim	Fuel limit	Fuel prec	Fuel limit	Fuel limit	Core tilt n	Core xenc	End-of-lif	Other fue	Core phys	Burnable

The letter ”e” in the example above denotes that VERT will use the exponential degradation modeling with the letter ”l” denoting linear. The secondary number is the ”b” value from the equations. Users can change the values to choose the degradation modeling they wish to include.

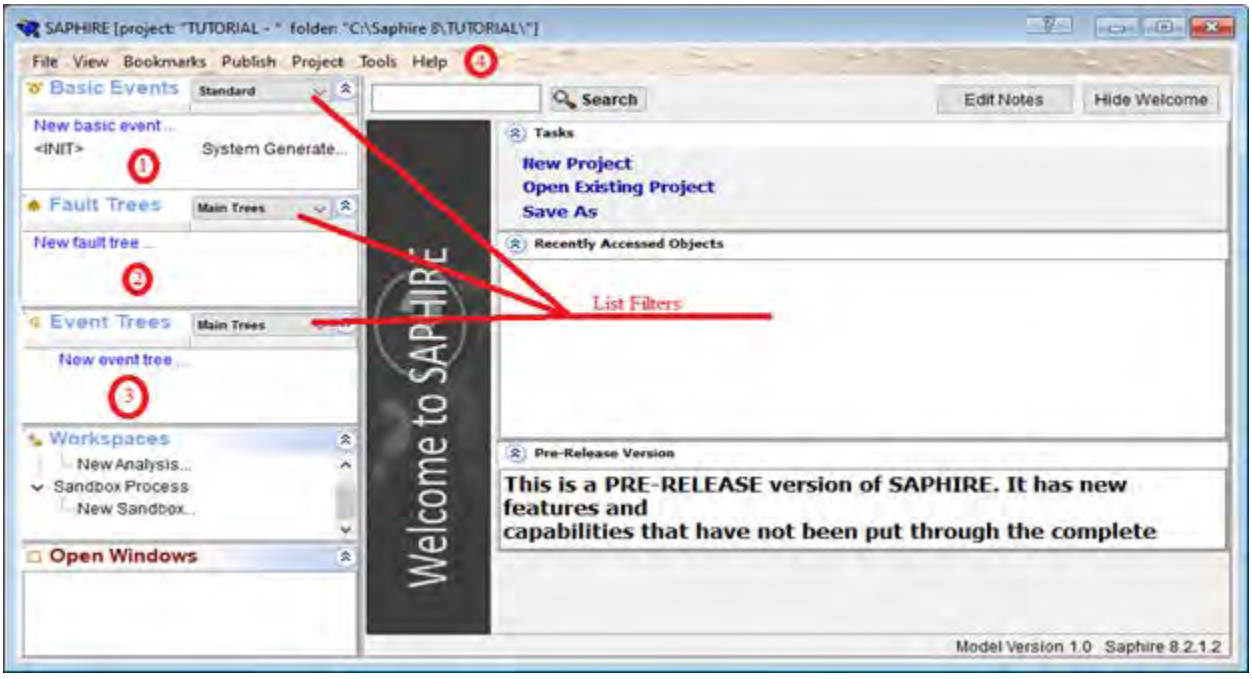
Changes regarding both SAPHIRE and RAVEN will be discussed in the following sections.

6 SAPHIRE

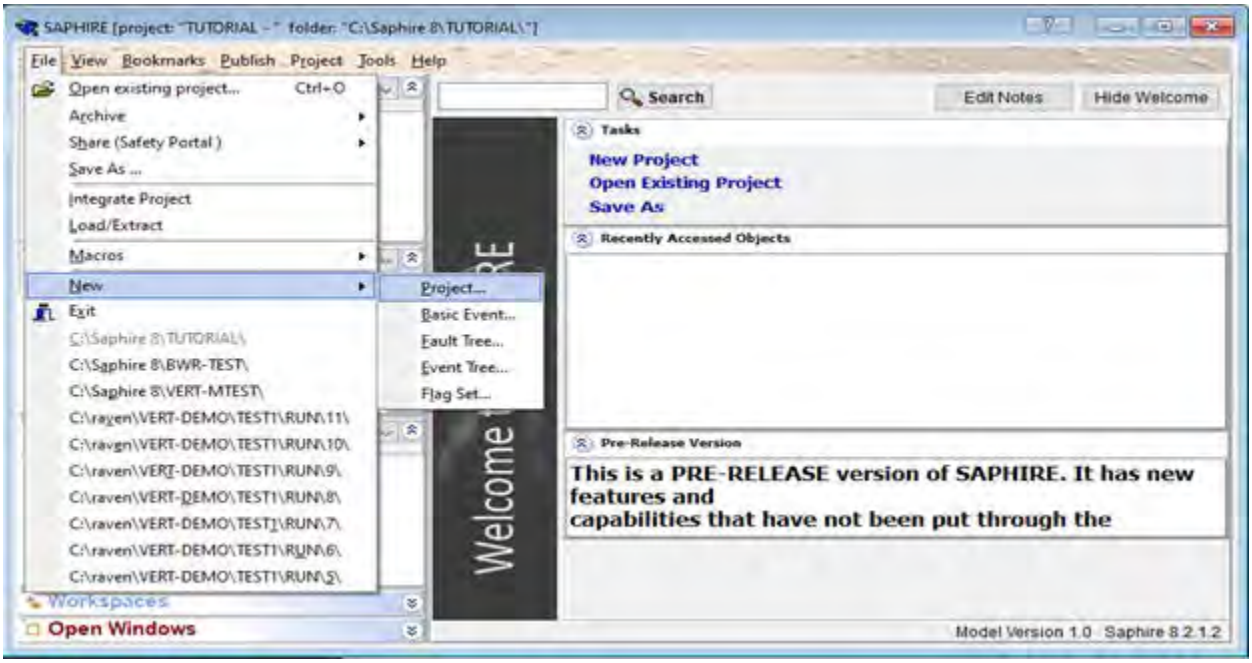
6.1 SAPHIRE Overview

This manual will cover the information necessary to use SAPHIRE in conjunction with VERT. This will also provide the reader with a basic tutorial of SAPHIRE itself. This will include how to navigate through SAPHIRE as well as creating and populating a project. For a more advanced explanation, please see the SAPHIRE user manual, SAPHIRE Version 8 Volume 3: User's Guide. VERT uses SAPHIRE for the framework of the basic events and fault trees. It will provide the user a custom post-processing script to run the fault trees. [5]

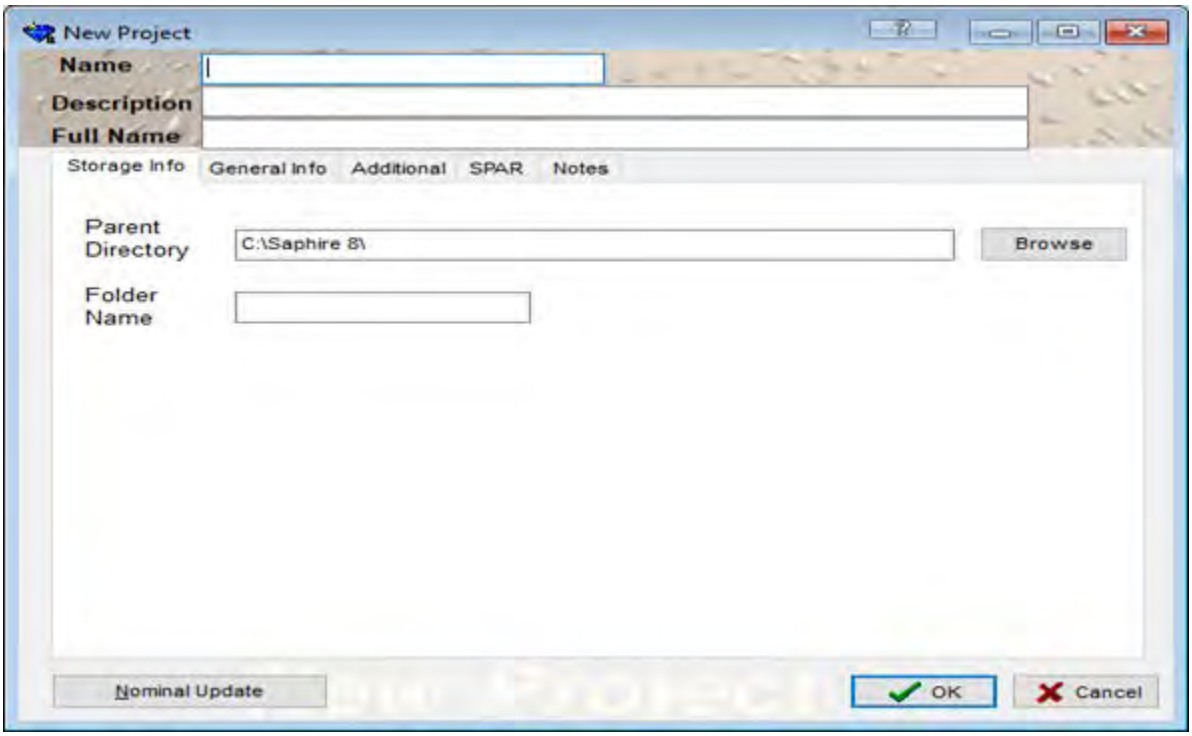
6.2 SAPHIRE Basics



This is the main window of SAPHIRE. The tool bar on the top (Number 4) contains the command lists for SAPHIRE. Window 1 contains the basic events list for the project; this can be resized and sorted according to the list selected from the list filter. Window 2 contains the fault trees for the project, and these will be made up from the basic events. Window 3 contains the event trees, which are made up of the fault trees. You can enter any chosen event, fault tree, or event tree by double clicking on it from the window. To create a new project, left click on "File," select "New," and click on "Project."



This will bring up the new project window. In the ”Storage Info” tab, the name field will be the file name of the project. The description is a tag to note what the project is. The full name will be displayed for published projects.



The general tab contains fields for the project creator, the creation date, version of the model, the default initiating event frequency units, site hazard curve, and any user-defined fields (e.g., facility, location, company). The additional tab has entries for the general philosophy behind the model, restricted use of the model, and warnings of the correct usage of the model. The SPAR tab has checkboxes to determine if it is a SPAR model or to use the SPAR-H adjustment equation listed on pg. 27 of NUREG/CR-6883. [5]

Once the project is created, SAPHIRE will store all the files in a project folder located where the user has determined. Inside, the folder subfolders are used to manage the data. [5]

- INIs - This folder stores .ini files related to project settings (e.g., bookmarks and general user settings).
- Mard - This folder stores any MAR-D files that are exported from SAPHIRE 8. MAR-D provides another interface to load or extract data files that define the PRA database. The files are in a ”flat-file” or ASCII file format. This is the legacy module for transferring files and is still fully supported in SAPHIRE 8.

  Typical uses of MAR-D include the:

  - Transfer of PRA information between data bases
    - Extracting MAR-D files from one SAPHIRE project and loading them (via MAR-D) into another SAPHIRE project. (The SAPHIRE project may be a new or previously existing project.)
  - Import of other PRA code information
    - Formatting the model information from another PRA code to use MAR-D file formats and creating a SAPHIRE project by loading the files via MAR-D.106.
  - Edit of PRA files using a text editor
    - Extracting MAR-D files from a SAPHIRE project, editing the files to make changes to the model or model descriptions, and loading those files (via MAR-D) back into the SAPHIRE project.
  - Archiving PRA files [5]
    - Saving the MAR-D files for long term storage in a text format rather than the native binary SAPHIRE format.
- Publish - This folder stores any report files that are exported from SAPHIRE 8
- Shared - This folder contains information that is shared among the main project and all workspaces. For example, project-related diagrams are stored in the ”Diagrams” subfolder. By sharing this information in a central place, the shared information is not duplicated each time a new workspace is created. This is also where the macro folder is located.

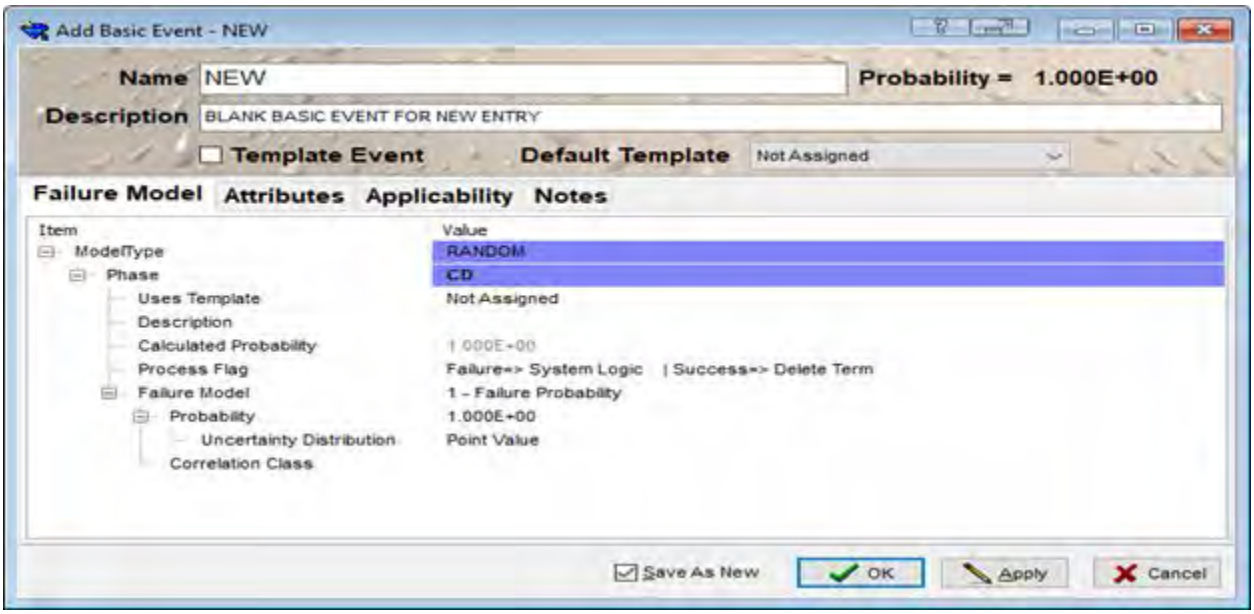
- Temp - This folder is used by SAPHIRE 8 during analysis. No project information files should be stored in this folder.
- Workspaces - This folder is used to store all workspaces that are saved. [5]

6.2.1 Basic Events

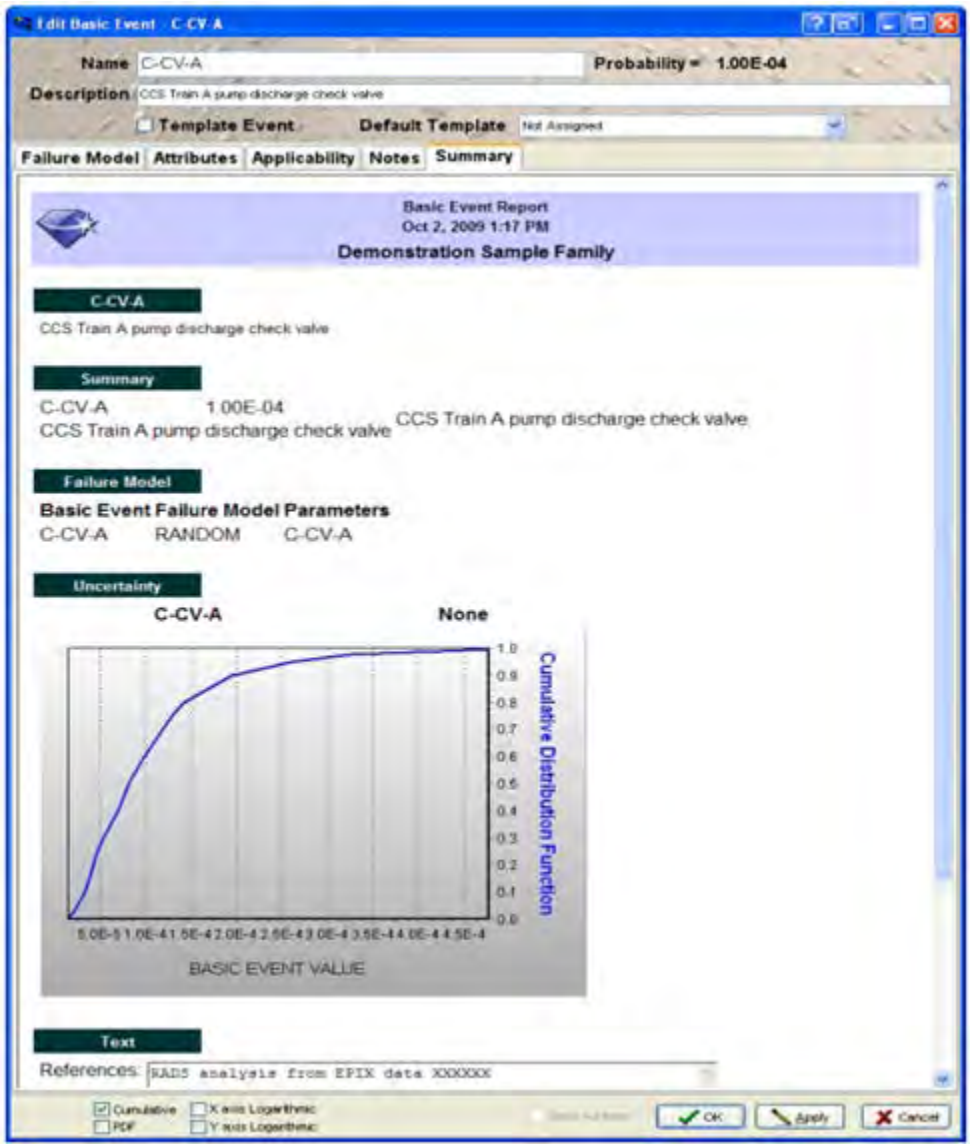
Unique terms used in SAPHIRE basic event editing are discussed below.

- Failure Model - SAPHIRE provides 14 unique calculation types to describe how the basic event succeeds or fails.
- Developed Event - A developed event is an event in SAPHIRE that is either an event tree top or fault tree gate. Regular fault tree basic events are not considered developed events.
- Delete Term - In SAPHIRE, the process known as ”delete term” refers to the removal of sequence success cut sets from the list of failure cut sets when generating sequence cut sets. As an example of the ”delete term,” consider a sequence where Top Event A is successful and Top Event B is failed. Any cut sets that would Fail A should not be allowed in the cut sets for B, so those cut sets are removed from the sequence [5].

To create a new basic event, double click on ”New Basic Event” in the basic event window or click on ”File,” select ”New,” and click on ”Basic Event.” This will bring up the ”Add Basic Event” window.



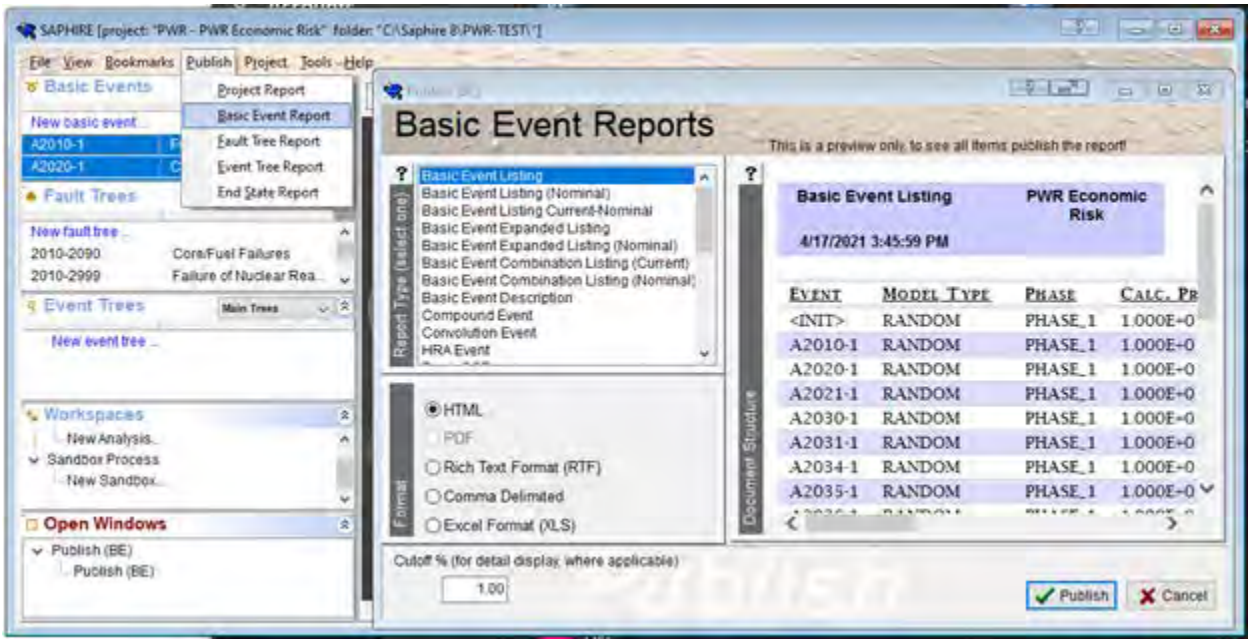
There are fields for the name and description of the event. Events can be set as a template so that a family of events can be changed simultaneously by changing the template. They can be assigned to a default template, which is listed from the user-created templates. The fields in the failure model window can be changed to fit the event being created by the user. [5] In the attributes tab, there are fields for an alternate name and alternate description. Drop-down lists for the event types and the event display types to designate the event. The applicability tab contains checkboxes for the user to determine what the event is applicable to. Once the event has been created, you can reenter into the basic event by right clicking on it from the event window and clicking "Edit Event" or by double clicking on the name. A new tab called summary will appear, which shows a summary of the event, including uncertainty, if the parameters have been specified. [5]



6.2.2 Generating a Basic Event Report*

this is supposed to be a sub-sub-sub-section;may have to change later

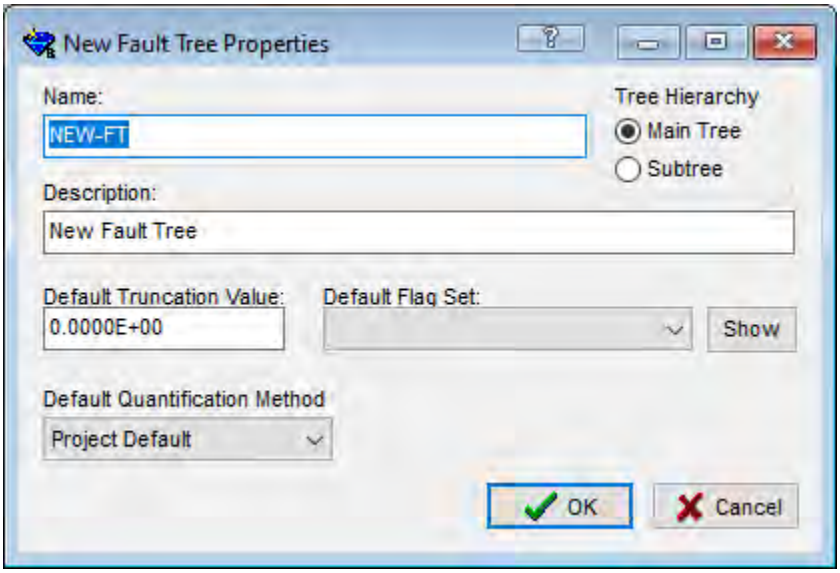
To generate a basic event report, select the basic event or events for SAPHIRE to generate the report on and left click on ”Publish.”



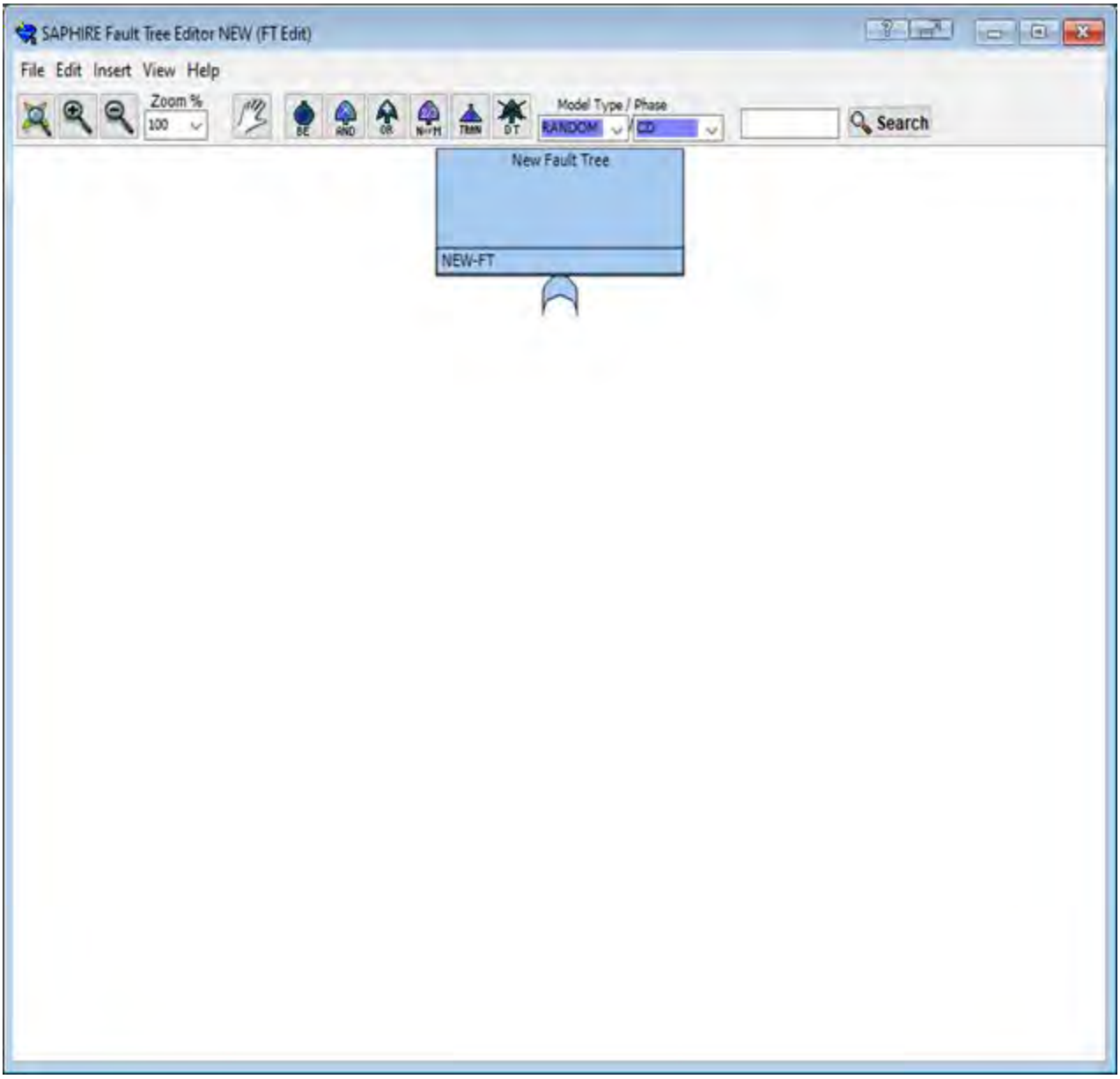
This will bring up the basic event reports window. The ”Report Type” allows the user to select the type of report they want to generate. The ”Format” window allows the user to choose the file format the report will generate in. The ”Document Structure” window previews what the report will look like in the selected format. The Cutoff Percentage allows the user to set a global cutoff percentage for detail display. Clicking on ”Publish” will generate the report.

6.2.3 Fault Trees

SAPHIRE enables the building of a fault trees through a graphical editor. A new fault tree can be constructed by double clicking on ”New Fault Tree” in the fault tree window or by left clicking on ”File,” selecting ”New,” and clicking on ”Fault Tree.” This will bring up the ”New Fault Tree” window.

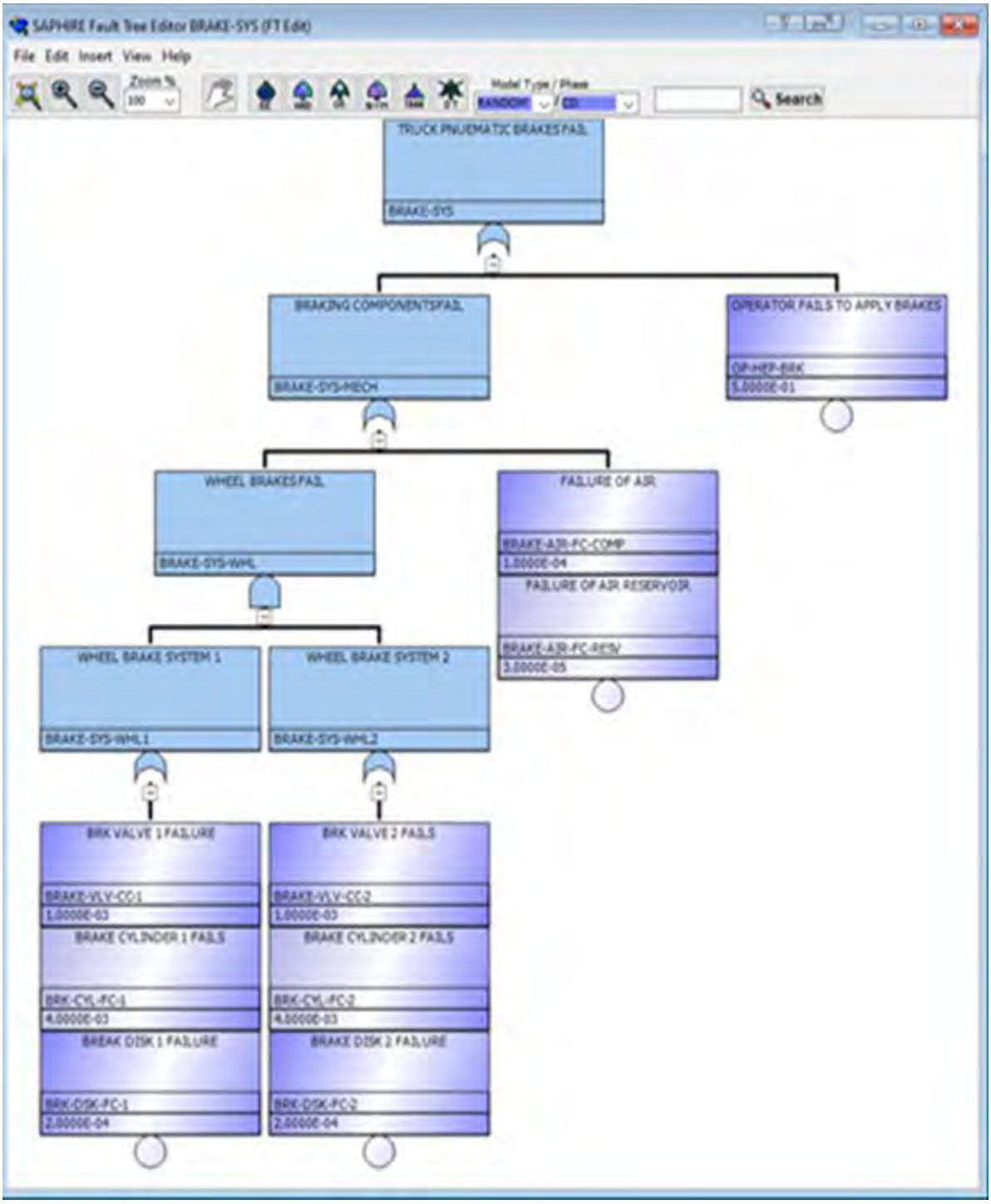


There is a field for the name and description. The user can set the default truncation value for the tree and a designation to set the tree as either a main tree or a subtree. The quantification method for the tree can be set here. Once the attributes have been set, clicking on ”OK” will bring up the graphical fault tree editor. [5]

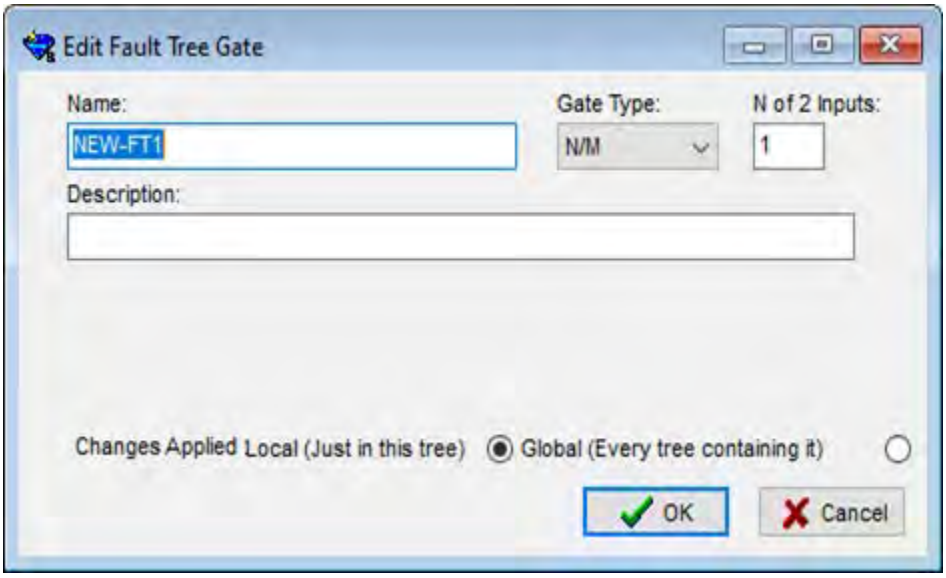


The magnifying glasses on the top left will allow the user to zoom in or our and move around the graphical area. At the top, the hand allows you to select and drag the event around and connect it to other gates or events. The ”BE” allows the creation of a new basic event. ”AND” creates an AND gate, ”OR” creates an OR gate, ”N-of-M” creates an N of M gate and ”DT” creates a delete term.

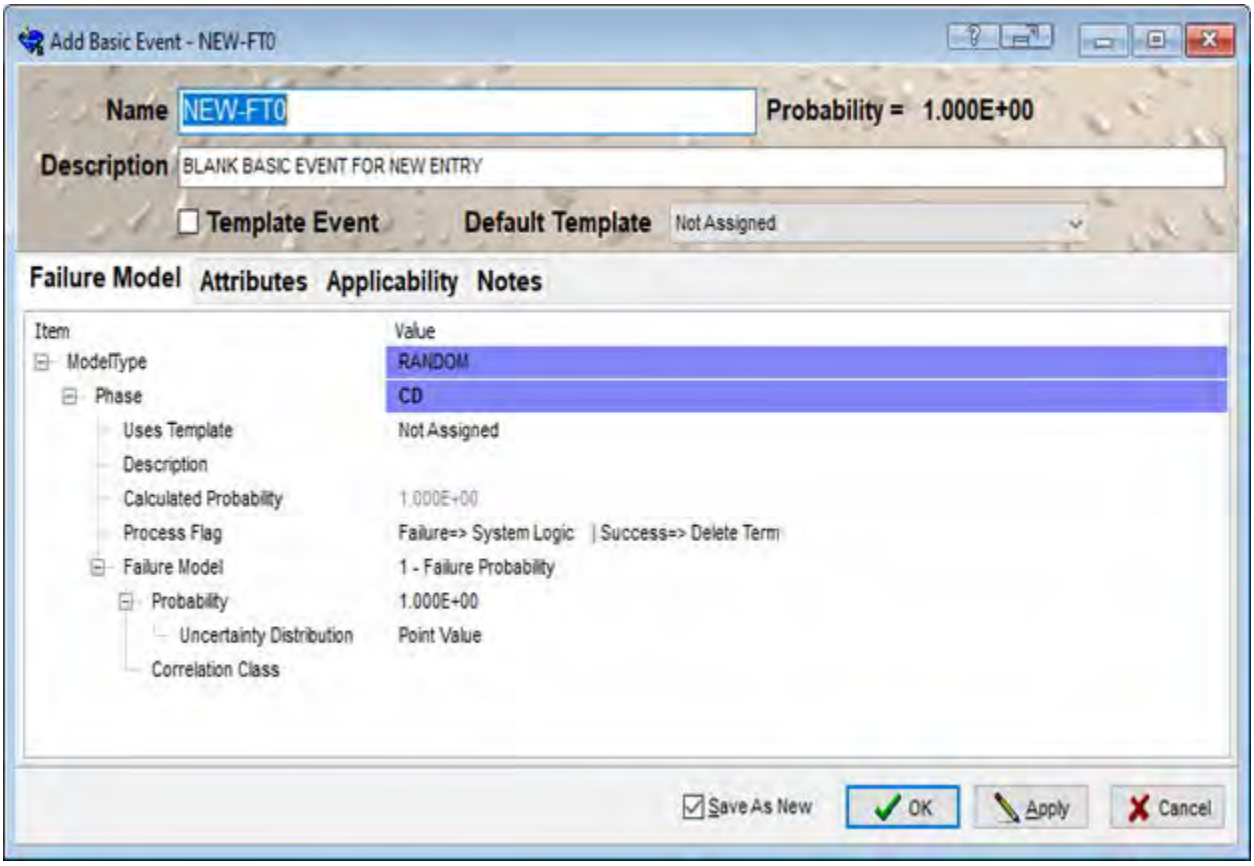
The figure below shows an example of a fault tree. The events are stacked, indicating they all flow into the same gate.



By double clicking on any of the gates, the user can change the attributes of that gate in the ”Edit Fault Tree Gate” window.

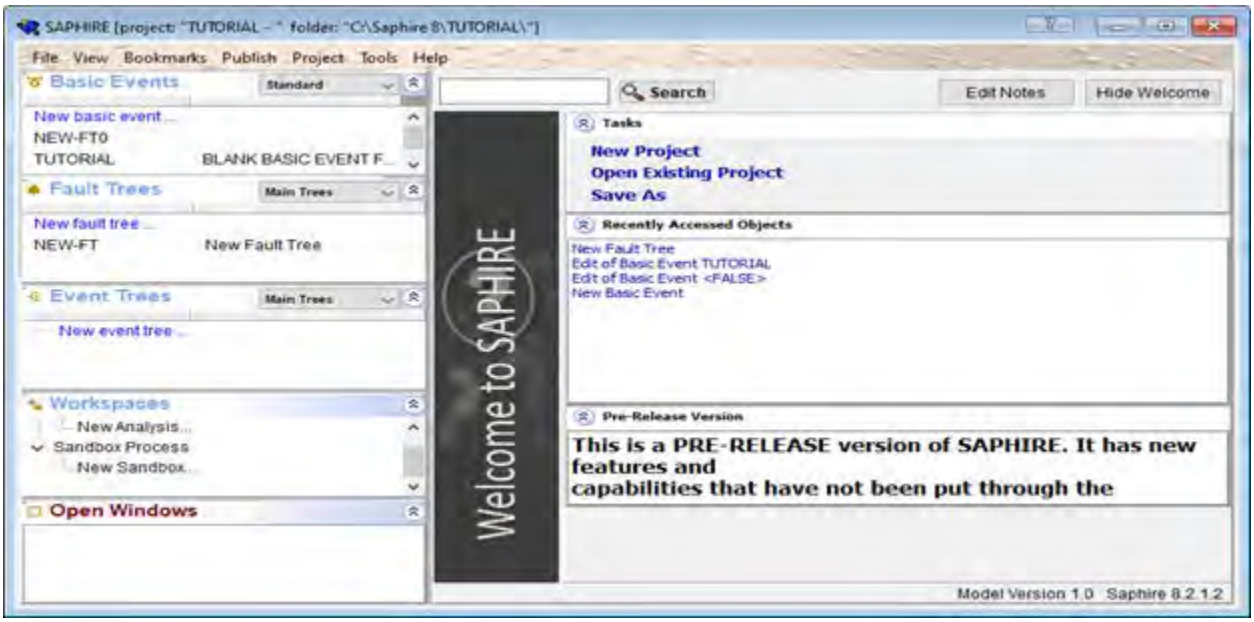


Double clicking on the basic event will bring up the basic event editor.



This basic event will be added to the basic event window and can also be edited there. To delete a

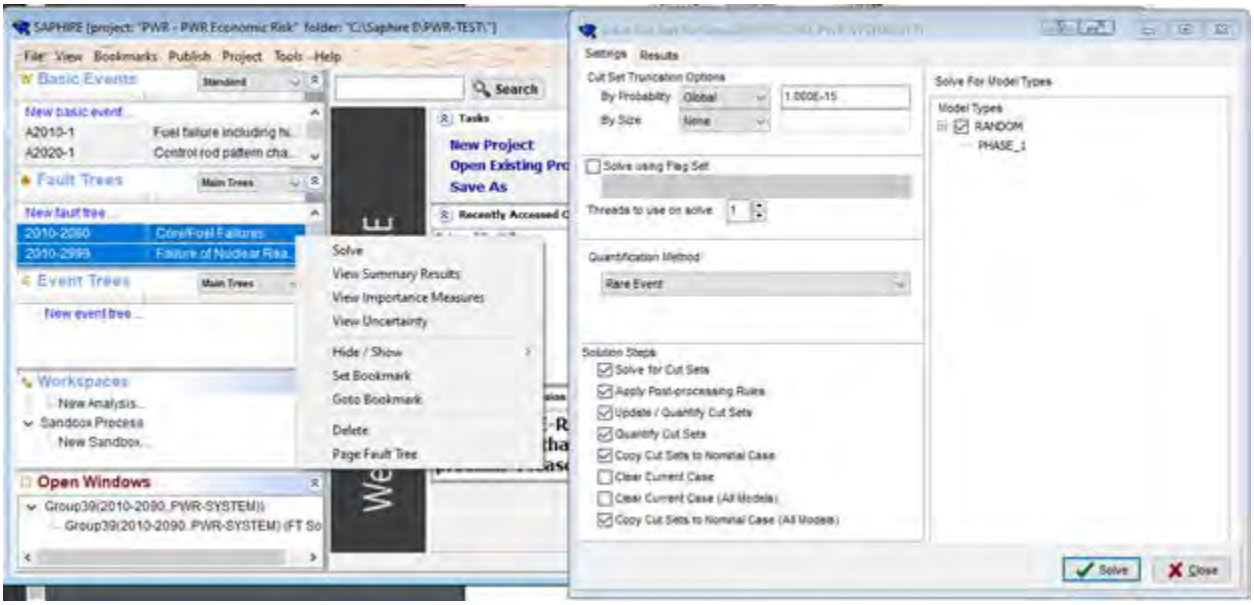
gate or basic event, right click on the gate and select ”Delete.” Once done with the fault tree, the user must save it by clicking on ”File” and selecting ”Save.” This will place the fault tree into the fault tree window on the main page.



6.2.4 Generating Fault Tree Cut Sets*

This is supposed to be a sub-sub-sub-section;may have to change later

To generate the cut sets for the fault tree, select one or more fault trees from the fault tree window. This will highlight them in blue. Right click on a highlighted fault tree, then select ”Solve.” This will bring up the ”Solve Cut Sets” window.



Select the desired truncation parameters and solution steps in the dialog box and choose ”Solve” to begin generating cut sets. [5]

To truncate solutions according to cut sets, locate the ”Probability Truncation” box and choose one of the following options:

- None - no truncation will be performed.
- Global - uses the cutoff value in the adjacent cut off field. This field defaults to the value set in the user settings of the project.
- System-specific - uses the cutoff value stored with the fault tree record (via the ”Edit Fault Tree” → ”Edit Properties” option).

To truncate solutions according to size, (this option is generally not used; ”Option” defaults to ”None”) the following options can be used:

- Size - Cut sets containing more items than a user-specified limit (specified in the adjacent text box) will be discarded.
- Zone - Cut sets having more zone flagged events than specified in the adjacent text box will be discarded.
- None - The number of events in a cut set will not affect whether the cut set is retained or discarded. This is the default option.

Check one or more of the following boxes to indicate which steps should be performed. Options that are not available (i.e., no recovery rules have been set up) will be grayed out. The steps will be performed in their logic order from top to bottom. [5]

- Solve for Cut Sets - If this box is checked, the fault tree logic will be solved for minimal cut sets using the truncation options specified.
- Apply Post-Processing Rules - If this box is checked, any post-processing rules associated with this fault tree will automatically be applied after the fault tree cut sets have been generated.
- Update/Quantify Cut Sets - When this box is checked, cut sets will be re-quantified using the truncation options specified. This option is useful for eliminating non-minimal cut sets introduced when recovery rules have been applied or to quickly re-quantify the cut sets when the basic event data have been altered.
- Copy Cut Sets to Nominal Case - When this box is checked, cut sets will be saved as the nominal case.

6.2.5 Event Trees

Event trees are logical representations of significant plant responses to initiating events with each sequence resulting in either a safe condition (such as a safe shutdown) or an accident condition (such as core damage). Event trees are developed by starting with an initiating event and branching to the right as various safety functions are questioned for success (up branch) or failure (down branch). Event trees provide a traceable way to perform the following functions. [5]

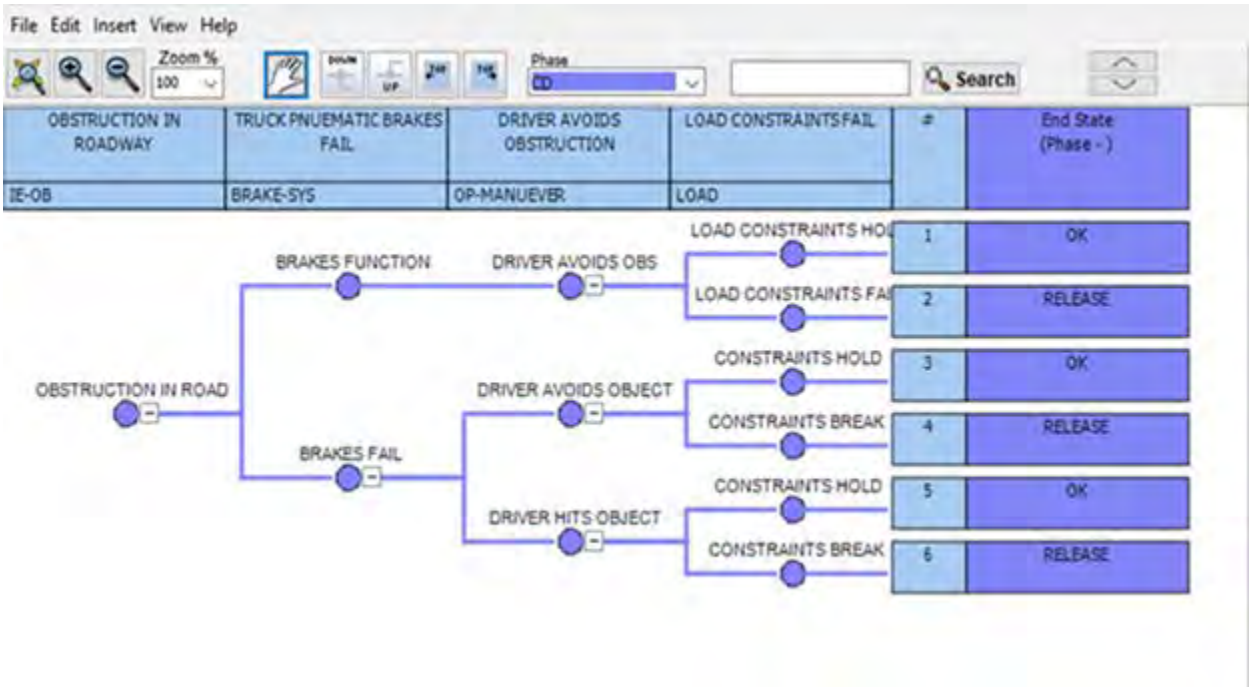
- Identify accident sequences
- Identify essential safety system functions
- Quantify sequence frequencies

Some key terms used in event tree development are [5]:

- Initiating Event - An initiating event is an operational occurrence (such as a loss-of-cooling accident or transient) that threatens fuel safety and may require a safety system response to avoid core damage.
- Top Event - Safety systems (or human actions) intended to respond to the initiating event.
- Branching - A branch underneath a top event indicates success with an up branch and failure with a down branch.

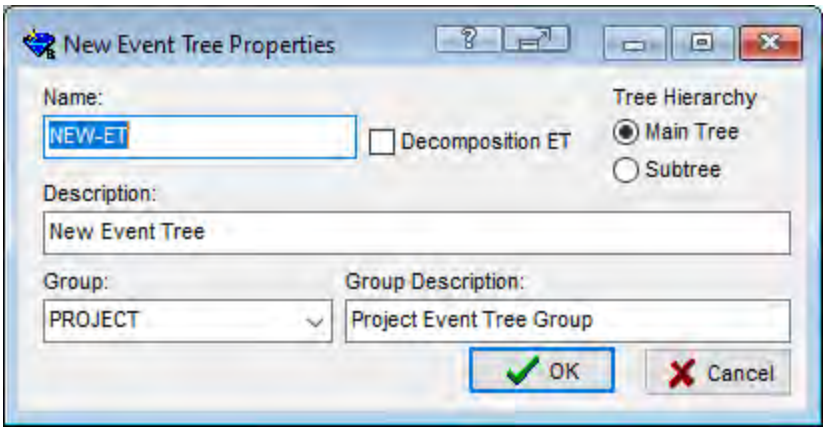
- Pass - When there is no branching beneath a top event, the top event is not relevant to the sequence.
- Sequence - This is the branching path, from initiating event to end state, which is a unique combination of system failures and successes.
- End State - This is a group of accident sequences that share certain characteristics, or outcomes, that the analyst delineates. These may be related to the ability to perform a safety function or the timing of functional failures.

The figure below shows an example of an event tree.



The boxes at the top represent the fault trees. Each circle is the conclusion of the fault tree with either a success or failure until the end state is reached.

To create a new event tree, either double click on the new event tree link in the event tree window or left click on ”File,” select ”New,” and click on ”Event Tree.” This will bring up the ”New Event Tree Properties” window.



There are fields for the event tree name, a description, and a group drop-down list. The event tree can be designated as a main tree or a subtree.

The event tree editor works the same way as the fault tree editor. To insert a new top event, click on one of the two ”Top” buttons located near the top of the event tree editor window. Another way to insert a top event is to left click on a top event in the event tree, select ”Insert,” and then select ”Before” or ”After.” To add a new branch, click on the branch you want to select and then either click on the ”Down” or ”Up” button located near the top of the event tree editor window. Alternatively, click on the branch, click on ”Insert,” and select either ”Above” or ”Below.” Once finished, the user must save the event tree by left clicking on ”File” and then ”Save.” The user will be able to access or edit the tree from the main window. [5]

6.2.6 Post Processing

This section presents the post-processing rules editors. These editors allow for the creation of rules that affect existing cut sets in a ”post-processing” fashion. The rule-based editors are available for both fault tree and sequence cut sets. [5]

Project-level post-processing rules applied to all event trees and fault trees are accessed from the main screen top menu bar ”Project Edit Rules ET (Post Processing)” or ”Project Edit Rules FT (Post Processing).” Rules applied to individual event trees and fault trees are accessed by right clicking on the appropriate tree and selecting ”Edit Cut Set Rules.” The SAPHIRE post-processing rules are ”free-form” logic rules that allow for the alteration or deletion of fault tree or sequence cut sets. [5]

The post-processing rules can be used for PRA techniques, such as [5]:

- The automated inclusion of sequence recovery events
- The inclusion of common-cause failure cut sets

- The elimination of mutually-exlusive events (e.g., impossible combinations of events).

The rules follow a format similar to the structure that is found in traditional programming languages (e.g., BASIC or PASCAL). As such, the ability exists to define ”macros” and ”if...then” type of structures. [5]

The post-processing rules may be developed for a particular fault tree, all fault trees, a particular sequence, a single event tree, or all sequences.

Item	Menu Path	Name of rule(s)
Specific fault tree	Fault Tree → Edit Cut Set Rules	“Fault Tree” Rule Level
All fault trees	Project → Edit Rules → FT	“Project” Rule Level
Single event tree	Event Tree → Edit Cut Set Rules	“Event Tree” Rule Level
All sequences	Project → Edit Rules → ET	“Project” Rule Level

The rules are entered in a free-form text editor within SAPHIRE. Note that the rules can be exported and loaded through MAR-D. The use of the post-processing rules could result in non-minimal cut sets. [5]

Thus, the typical steps in performing an analysis using the post-processing rules are [5]:

1. Finalize logic models and data changes by saving any modified trees.
2. Solve fault tree or sequence cut sets.
3. Apply post-processing rules to applicable fault trees or sequences.
4. Perform a cut set update to fault tree or sequence cut sets.

The rule editor searches existing fault trees or sequence cut sets for cut sets matching the search criteria defined in the rule. The rule is used to modify the cut sets matching the search criteria. [5]

Symbols

|	Denotes a comment line	~	Operator for "not present"
*	Logical AND operator	+	Logical OR operator
/	Complement	( )	Parentheses

Search Criteria	Meaning of the Search Criteria
X	Basic event X appears in the cut set
~X	Basic event X does not occur in the cut set
/X	Success of basic event X appears in the cut set
X * Y	Both basic events X and Y appear in the cut set
X + Y	Either basic event X or Y appear in the cut set
X*(Y + Z)	Either X and Y or X and Z appear in cut set in the cut set
~X*Y	Basic event Y does appear and basic event X does not appear
always	This pre-defined macro-name means the criteria is always met.
system(ECS)	Fault tree top event with name ECS

The table below shows an example of the ”if-then-elsif” structure (the example below deletes the cut set if both diesel generators are out for maintenance. If the two diesel generators fail randomly, add a common-cause event) [5]:

if (DG-1-MAINT*DG-2-MAINT) then
DeleteRoot;
elseif (DG-1-RAND*DG-2-RAND) then
(copy the original cut set, remove the two failure events, then add CC CopyRoot;)
DeleteEvent=DG-1-RAND; DeletEvent=DG-2-RAND;
AddEvent=DG-CCF-1AND2;
Endif

The ”AddEvent” keyword can be selected from the right side of the rules editor and denotes that an event will be added to the cut set being evaluated.

The ”DeleteEvent” keyword indicates that an event will be deleted from the cut set being evaluated.

The ”DeleteRoot” keyword indicates that the original cut set (i.e., the cut set that satisfied the search criteria) will be deleted.

6.3 VERT Modifications in SAPHIRE

The generic models of the pressurized-water reactor and boiling-water reactor come with basic events loaded with probabilities and cause codes gained from the GADS data. The fault trees follow a generic power plant sub-system. Users can build and link event trees to the fault trees to run scenarios for the specified event. These can be updated to reflect information the user wishes to model within VERT following the guide above. The user may also change the post-processing script in the script editor to reflect an individual project. If more details are needed to modify the SAPHIRE project, please see SAPHIRE Version 8 Users' Guide

7 RAVEN

7.1 RAVEN Overview

This manual will provide a basic understanding of VERT's usage of RAVEN. This will include the basics of the RAVEN script in XML format and the distribution and sampling methods used by VERT, along with covering the necessary SAPHIRE interface. VERT will have RAVEN complete multiple runs through the SAPHIRE interface to iterate on the models in a time-dependent manner. This interface will also be covered in the manual.

7.2 Input Structure

The RAVEN code does not have a fixed calculation flow, since all of its basic objects can be combined in order to create a user-defined calculation flow. Thus, its input (XML format) is organized in different XML blocks, each with a different function. The main input blocks are [2]:

- <Simulation>: The root node containing the entire input; all of the following blcoks fit inside the "Simulation" block
- <RunInfo>: Specifies the calculation settings (number of parallel simulations, etc.)
- <Files>: Specifies the files to be used in the calculation
- <Distributions>: Defines disributions needed for describing parameters, etc.
- <Samplers>: Sets up the strategies used for exploring an uncertain domain
- <Optimizers>: Sets up the strategies for minimizing/maximizing an objective function
- <DataObjects>: Specifies internal data objects used by RAVEN
- <Databases>: Lists the HDF5 databases used as input/output to a RAVEN run
- <OutStreams>: Visualization and printing system block
- <Models>: Specifies codes, ROMs, post-proessing analysis, etc.
- <Functions>: Details interfaces to external user-defined functions and modules, and interfaces the user will be building and/or running
- <Steps>: Combines other blocks to detail a step in the RAVEN workflow, including I/O and computations to be performed

Each block within RAVEN also makes use of a verbosity system, which allows a user to control the level of output sent to the user interface. These settings are declared globally as attributes in the node and locally in each block. VERT currently uses a debug verbosity where all errors, warnings, and debug messages are displayed. [2]

The nodes below are the XML nodes used within VERT:

- <WorkingDir>, string, required field, specifies the absolute or relative (with respect to the location where the .xml file is located) path to a directory that will store all the results of the calculations and where RAVEN looks for the files specified in the <Files> block. If "runRelative='True'" is used as an attribute, it will be relative to where RAVEN is run.
- <batchSize>, integer, optional field, specifies the number of parallel runs executed simultaneously (e.g., the number of driven code instances, such as RELAP5-3D, that RAVEN will spawn at the same time). Each parallel run will use NumThreads * NumMPI cores.
- <Sequence>, comma-separated string, a required field, is an ordered list of the step names that RAVEN will run.

The <Files> block defines any files that might be needed within the RAVEN run. This could include inputs to the model, pickled ROM files, or CSV files for post-processors, to name a few. Each entry in the <Files> block is a tag with the file type. Files given through the input XML at this point are all <Input> type. [2]

- name, required string attribute, user-defined name of the file. This does not need to be the actual filename; this is the name by which RAVEN will identify the file. Note that as with other objects, this name can be used to refer to this specific entity from other input blocks in the XML.

VERT uses a uniform distribution, which is a continuous distribution with a rectangular-shaped probability density function. It is often used where the distribution is only vaguely known, but upper and lower limits are known. The specifications of this distribution must be defined within a <Uniform> XML block. This XML node accepts one attribute [2]:

- name, required string attribute, user-defined name of this distribution. Note that as with other objects, this identifier can be used to reference this specific entity from other input blocks in the XML.
- <lowerBound>, float, required parameter, domain lower boundary.
- <upperBound>, float, required parameter, domain upper boundary.

Note that since the uniform distribution is a rectangular-shaped probability density function, the truncation does not have any effect; this is why the children nodes are the ones generally used for truncated distributions.

The sampler is probably the most important entity in the RAVEN framework. It performs the driving of the specific sampling strategy and, hence, determines the effectiveness of the analysis, from both an accuracy and computational point of view. VERT uses the grid sampling approach. The idea is to construct an N-dimensional grid where each dimension is represented by one uncertain variable. This approach performs the sampling at each node of the grid. The sampling of the grid consists of evaluating the answer from the system under all possible combinations among the different variables' values with respect to a predefined discretization metric. In RAVEN, two discretization metrics are available: cumulative distribution function and value. Thus, the grid meshing can be input via probability or variable values. Regarding the N-dimensional distributions, the user can specify for each dimension the type of grid to be used (i.e., value or cumulative distribution function). Note that the discretization of the cumulative distribution function, for only the grid sampler, is performed on the marginal distribution for the specific variable considered. [2]

In the <Grid> input block, the user needs to specify the variables to sample. As already mentioned, these variables are specified within consecutive <variable> XML blocks:

- <variable>, XML node, required parameter can specify the following attributes:
  - name, required string attribute, user-defined name of this variable
  - shape, comma-separated integers, optional field, determines the number of samples and shape of samples to be taken. For example, shape="2,3" will provide a 2 x 3 matrix of values, while shape="10" will produce a vector of 10 values. Omitting this optional attribute will result in a single scalar value instead. Each of the values in the matrix or vector will be the same as the single sampled value. Note that a model interface must be prepared to handle non-scalar inputs to use this option. [2]

This <variable> recognizes the following child nodes [2]:

- <distribution>, string, required field, name of the distribution that is associated to this variable. Its name needs to be contained in the <Distributions> block. In addition, if "NDDistribution" is used, the attribute "dim" is required. Alternatively, this node must be omitted if the <function> is supplied.
- <function>, string, required field, name of the function that defines the calculation of this variable from other distributed variables. Its name needs to be contained in the <Functions> block. This function must implement a method named "evaluate." Alternatively, this node must be omitted if the <distribution> node is supplied.

- <grid>, space separated floats, required field, the content of this XML node depends on the definition of the associated attributes:
  - type, required string attribute, user-defined discretization metric type: 1) ”CDF”, the grid will be specified based on cumulative distribution function probability thresholds, and 2) ”value”, the grid will be provided using variable values.
  - construction, required string attribute, how the grid needs to be constructed, independent of its type (i.e., ”CDF” or ”value”).

Based on the construction type, the content of the <grid> XML node and the requirements for other attributes change [2]:

- construction=”equal”. The grid is going to be constructed equally spaced (type=”value”) or equally probable (type=”CDF”). This construction type requires the definition of additional attributes.
- steps, required integer attribute, number of equally spaced or probable discretization steps.

This construction type requires that the content of the <grid> node represent the lower and upper bounds (either in probability or value). Two values need to be specified; the lowest one will be considered as the ”lowerBound” and the largest, the ”upperBound.” The lower and upper bounds are checked against the associated <distribution> bounds. If one or both of them falls outside the distribution’s bounds, the code will raise an error. The ”stepSize” is determined by ”stepSize=(upperBound-lowerBound)/steps. [2]

```
<Distributions>
    <Uniform name="time">
        <lowerBound>0</lowerBound>
        <upperBound>2.0</upperBound>
    </Uniform>
</Distributions>
```

For using the SAPHIRE interface within RAVEN, all the files needed for the code to run should be specified. The files typically needed are [2]:

- SAPHIRE macro input file with file extension ’.mac’.
- SAPHIRE compressed project inputs with file extension ’.zip’.

```
<Files>
    <Input name="macro" type="">C:\Saphire 8\VERT-MTEST\Shared\Macros\VERT-MTEST.mac</Input>
    <Input name="project" type="">C:\Saphire 8\VERT-MTEST.zip</Input>
</Files>
```

In the <models> block, the SAPHIRE executable needs to be specified. The figure below is an example from the demonstration.

```
<Models>
    <Code name="saphire" subType="Saphire">
        <shell>False</shell>
        <executable>"C:\Saphire 8\tools\SAPHIRE.exe"</executable>
        <clargs arg="macro" extension=".mac" type="input" delimiter="="/>
        <clargs arg="project" extension=".zip" type="input" delimiter="="/>
        <outputFile>fixed_output.csv</outputFile>
        <codeOutput type="uncertainty">ft_uq.csv</codeOutput>
    </Code>
</Models>
```

The <Code> XML node contains the information needed to execute the specific external code. This XML node accepts the following attributes [2]:

- name, required string attribute, user-defined identifier of this model. Note that as with other objects, this identifier can be used to reference this specific entity from other input blocks in the XML.
- subType, required string attribute, specifies the code that needs to be associated with this model.

This model can be initialized with the following children nodes [2]:

- <executable>, string, required field, specifies the path of the executable to be used.
- <clargs>, string, required field, allows the addition of command-line arguments to the execution command. This node is used to specify the input files that are required by SAPHIRE. This node accepts the following attributes:
  - type, string, required attribute, specifies the type of command-line argument to add. The current option is ”input”.
  - arg, string, required field, specifies the flag to be used before the entry.

  - extension, string, required field, specifies the type of file extension to use. This links the <Input> file in the <Steps> to this location in the execution command. Currently only accepts ”.zip” and ”.mac”.
  - delimiter, string, required field, used to link the ”arg” and the <Input> with the extension given by ”extension”.

- <outputFile>, string, optional field, used to specify the output file name (CSV only). In this case, the code interface always produces a CSV file named ”fixed output.csv”
- <codeOutput>, string, optional field, used to specify the output file generated by SAPHIRE that will be processed via the code interface. The following attributes can be specified:
  - type, string, required attribute, the actual type of the provided file. The only type accepted here is ”uncertainty”.

The figure below is an example of the XML file generated in the demonstration from earlier.

TEST.xml - Notepad

File Edit Format View Help

```
<?xml version="1.0"?>
<Simulation verbosity="debug">
  <RunInfo>
      <Sequence>RUN,PLOT</Sequence>
      <WorkingDir>TEST1</WorkingDir>
      <batchSize>1</batchSize>
  </RunInfo>

  <Files>
      <Input name="macro" type="">C:\Saphire 8\VERT-MTEST\Shared\Macros\VERT-MTEST.mac</Input>
      <Input name="project" type="">C:\Saphire 8\VERT-MTEST.zip</Input>
  </Files>

  <Models>
      <Code name="saphire" subType="Saphire">
          <shell>False</shell>
          <executable>"C:\Saphire 8\tools\SAPHIRE.exe"</executable>
          <clargs arg="macro" extension=".mac" type="input" delimiter="="/>
          <clargs arg="project" extension=".zip" type="input" delimiter="="/>
          <outputFile>fixed_output.csv</outputFile>
          <codeOutput type="uncertainty">ft_uq.csv</codeOutput>
      </Code>
  </Models>

  <Distributions>
      <Uniform name="time">
          <lowerBound>0</lowerBound>
          <upperBound>2.0</upperBound>
      </Uniform>
  </Distributions>

  <Samplers>
      <Grid name="timeSpace">
          <variable name="t">
              <distribution>time</distribution>
              <grid construction="equal" steps="10" type="CDF">0.0 1.0</grid>
          </variable>
          <variable name="2070">
                <function>2070</function>
          </variable>
          <constant name="p2070">4.6e-06</constant>
```

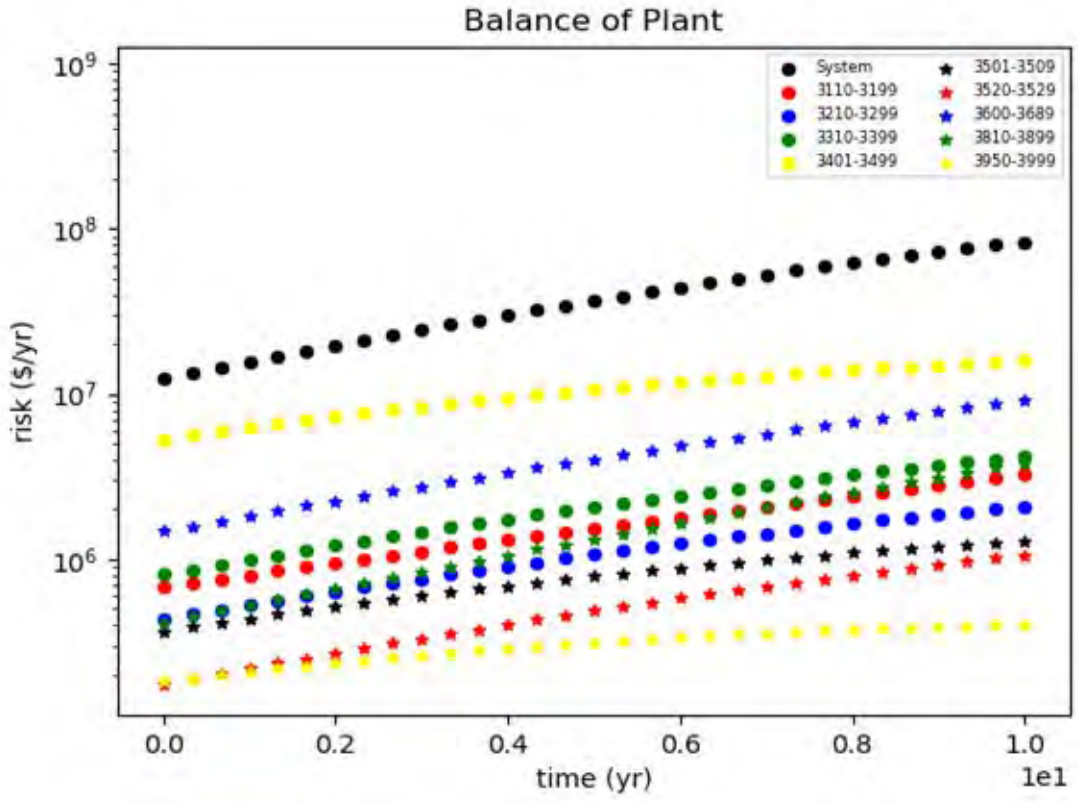
Ln 13, Col 1 100% Windows (CRLF) UTF-8

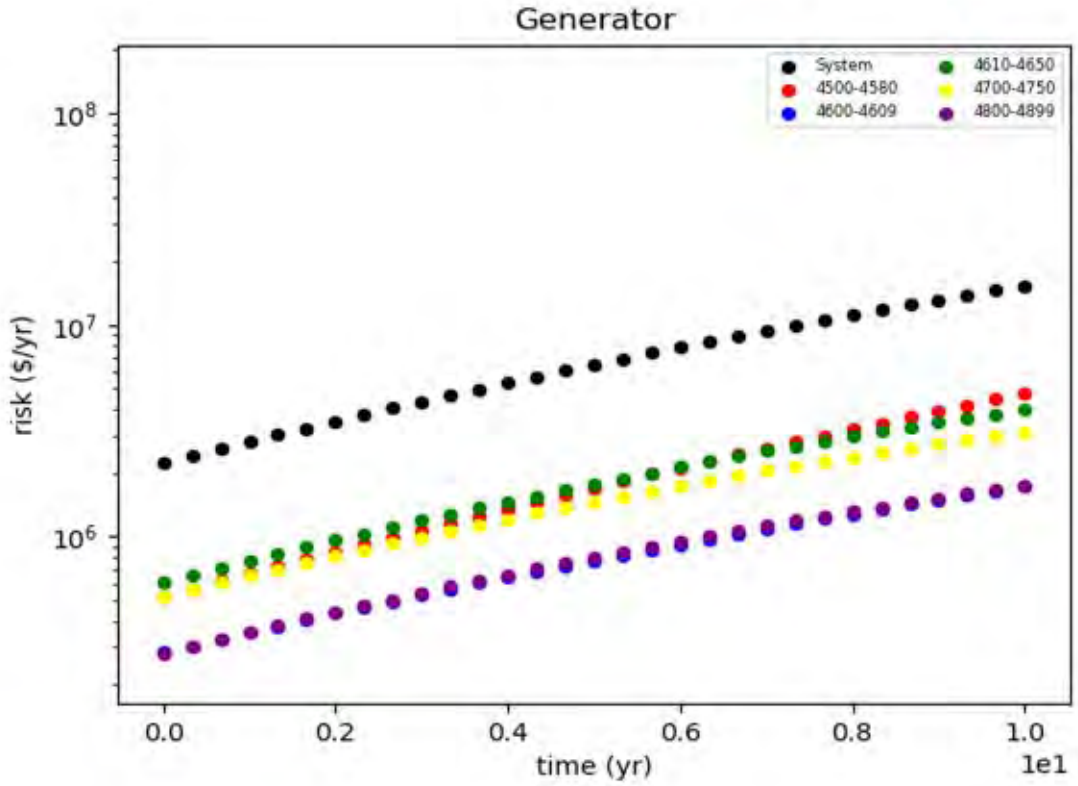
8 Test Cases and Results

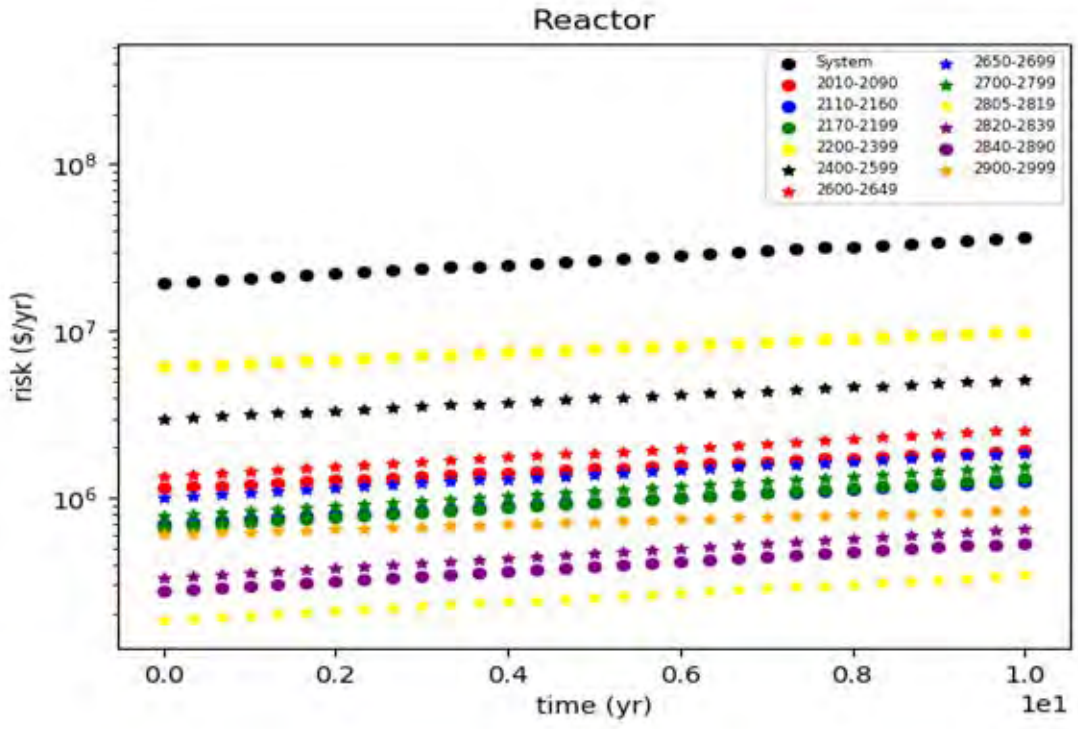
8.1 Overview

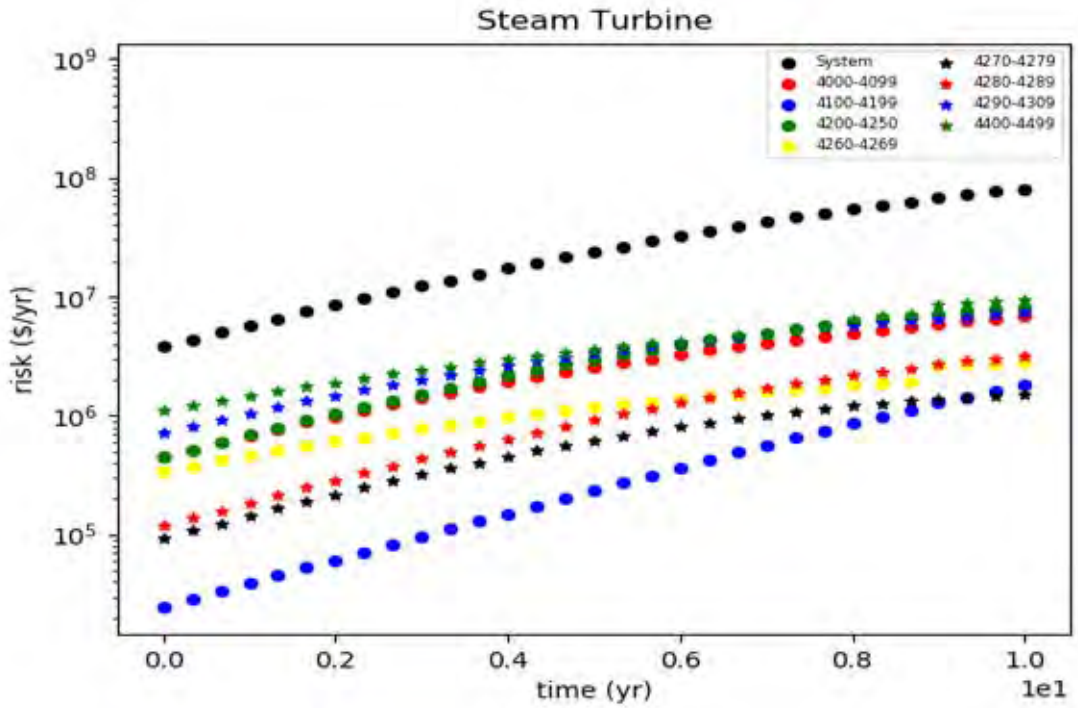
This section will present two test cases of a generic boiling-water reactor. The first was calculated over a 10-year period using 30 time steps. The cost per megawatt-hour for electricity was set as $33.50 and the power output was set as 1,000 MWe. The second test case models the behavior if the cause code 3410 in the feedwater sub-system was replaced by a component with a reliability an order of magnitude higher than the previous component and with a variable cost per megawatt-hour and variable power output. The cost per megawatt-hour was set as follows: $127 Jan.-Mar., $133 Apr.-Jun., $132 Jul.-Sep., and $128 Oct.-Dec. The power output was set as follows: 948 MWe Jan.-Mar., 906 MWe Apr.-Jun., 963 MWe Jul.-Sep., and 920 MWe Oct.-Dec. over a 2-year period with 20 time steps. [7]

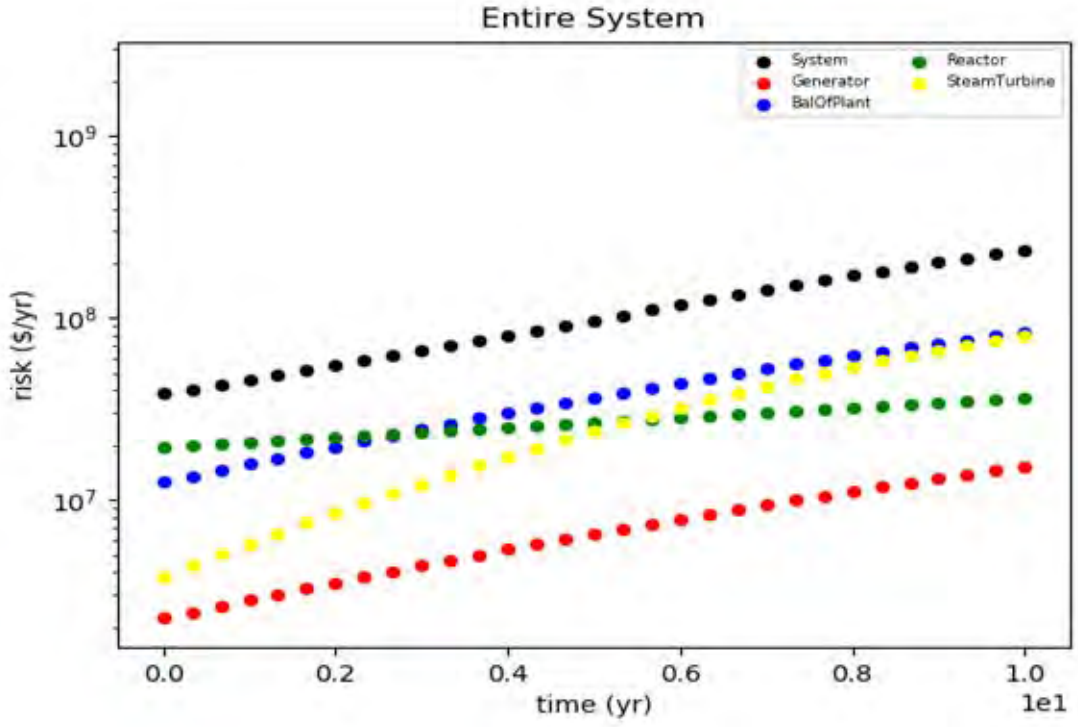
8.2 BWR



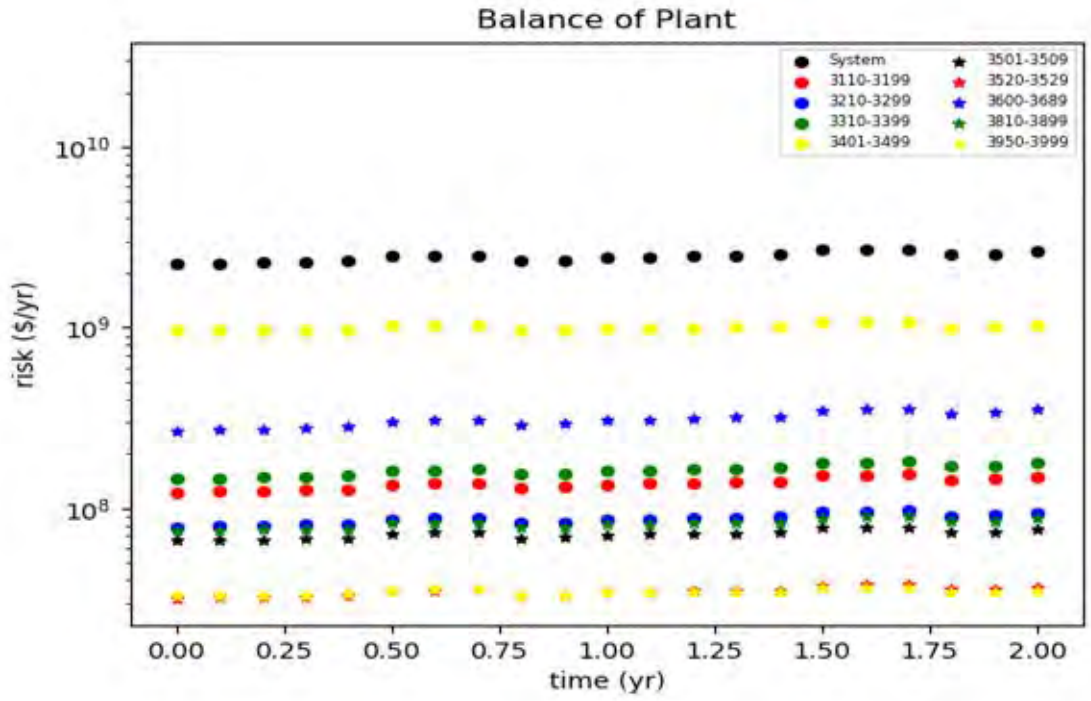

Generator
System
4500-4580
4600-4609
4610-4650
4700-4750
4800-4899
risk ($/yr)
time (yr)
1e1

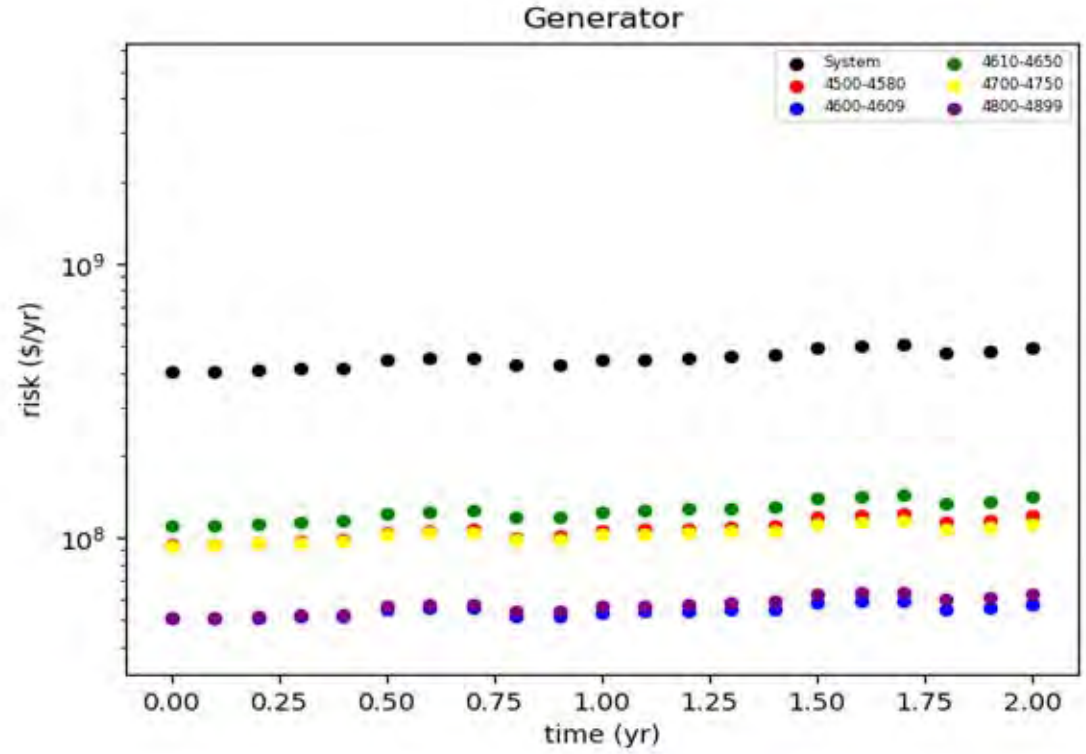

Reactor
System
2010-2090
2110-2160
2170-2199
2200-2399
2400-2599
2600-2649
2650-2699
2700-2799
2805-2819
2820-2839
2840-2890
2900-2999
risk ($/yr)
time (yr)
1e1

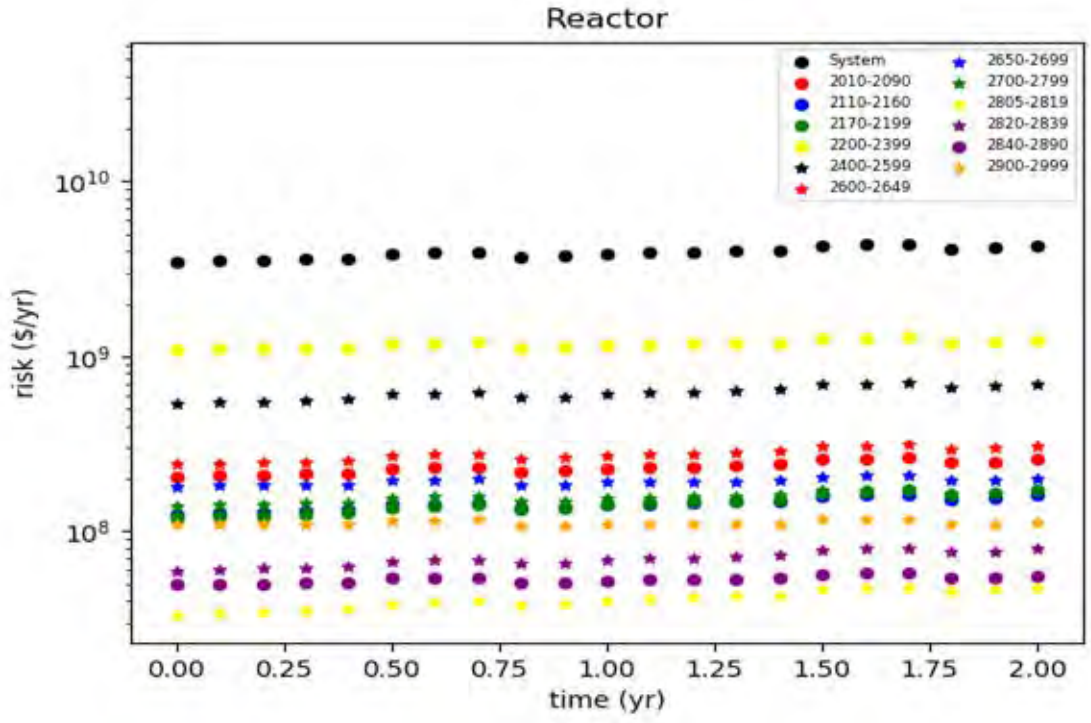

Steam Turbine
System
4000-4099
4100-4199
4200-4250
4260-4269
4270-4279
4280-4289
4290-4309
4400-4499
risk ($/yr)
time (yr)
1e1

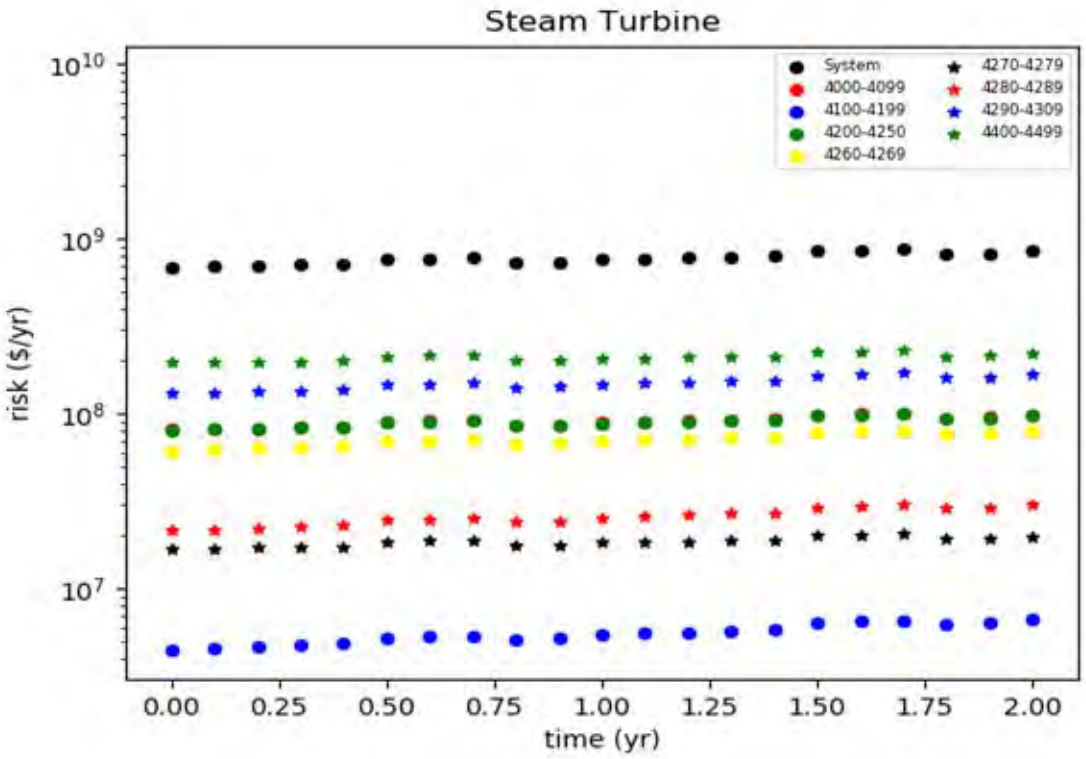

Entire System
System
Generator
BalOfPlant
Reactor
SteamTurbine
risk ($/yr)
time (yr)
1e1

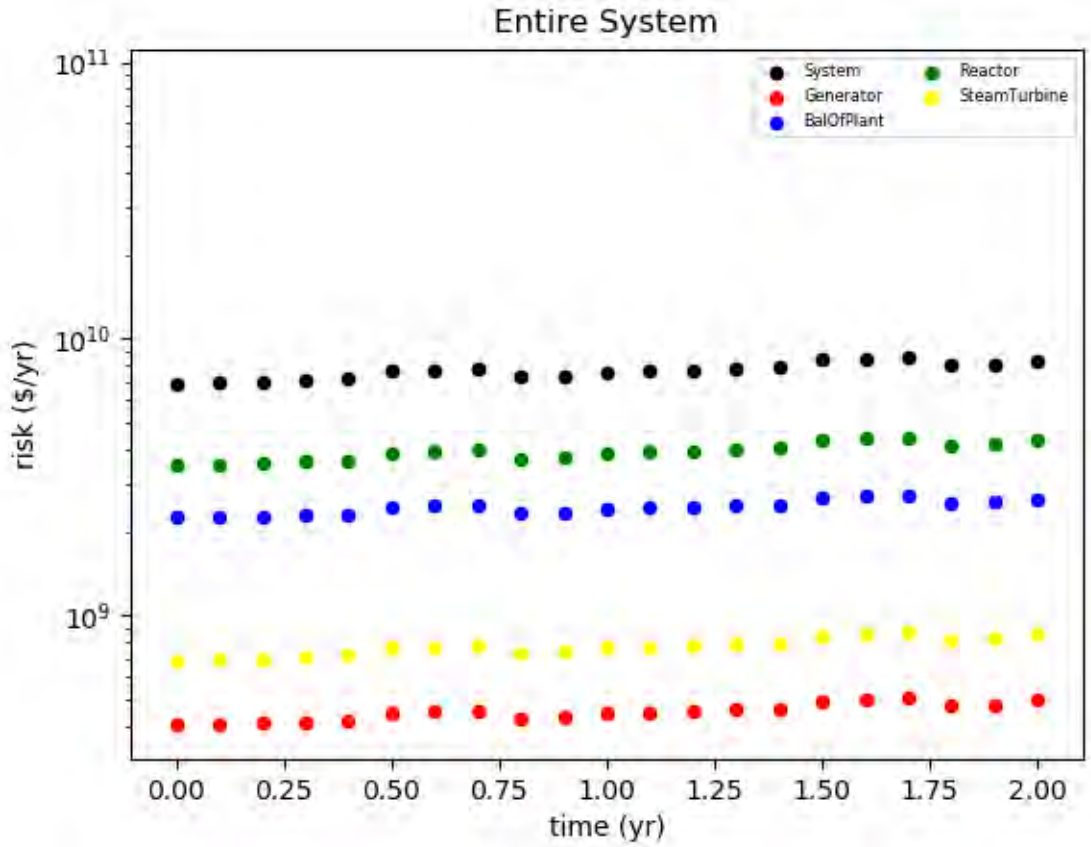
8.3 BWR (Upgraded)




Generator
System
4500-4580
4600-4609
4610-4650
4700-4750
4800-4899
risk ($/yr)
10^9
10^8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)


Reactor
System
2010-2090
2110-2160
2170-2199
2200-2399
2400-2599
2600-2649
2650-2699
2700-2799
2805-2819
2820-2839
2840-2890
2900-2999
risk ($/yr)
10^{10}
10^9
10^8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)


Steam Turbine
System
4000-4099
4100-4199
4200-4250
4260-4269
4270-4279
4280-4289
4290-4309
4400-4499
risk ($/yr)
time (yr)
10^{10}
10^{9}
10^{8}
10^{7}
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Entire System
System
Generator
BalOfPlant
Reactor
SteamTurbine
10^{11}
10^{10}
10^{9}
risk ($/yr)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
time (yr)

9 Appendix

Document Version Information

This document has been compiled using the following version of the plug-in git repository:

References

[1] A. Alfonsi, C. Rabiti, D. Mandelli, J. Cogliati, C. Wang, P. Talbot, D. Maljovec, and C. Smith, "Raven user guide," Tech. Rep. INL/EXT-18-44465, April 2020.

[2] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, and J. Chen, "Raven user manual," Tech. Rep. INL/EXT-15-34123, March 2017.

[3] "Miniconda."

[4] A. Alfonsi, C. Wang, and J. Chen, "title of page."

[5] C. Smith and S. Wood, "Systems analysis programs for hands-on integrated reliability evaluations (saphire) version 8 volume 3: User's guide," Tech. Rep. NUREG/CR-7039.

[6] J. Miller and S. Ercanbrack, "Vert tests using a demo model."

[7] J. Miller, S. Ercanbrack, and C. Pope, "The versatile economic risk tool (vert),"