DISCLAIMER:

This document does not meet the
current format guidelines of
the Graduate School at
The University of Texas at Austin.

It has been published for
informational use only.

The Dissertation Committee for Robert Ryan Flanagan Certifies that this is the approved version of the following dissertation:

Novel Methods for Generalizing Nuclear Fuel Cycle Design, and Fuel Burnup Modeling

Committee:

Erich Schneider, Supervisor

Steven Biegalski

Sheldon Landsberger

Paul Wilson

Yarden Livnat

Novel Methods for Generalizing Nuclear Fuel Cycle Design, and Fuel Burnup Modeling

by

Robert Ryan Flanagan M.S.E. B.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2015

Dedication

I'd like dedicate this work to my parents; Hugh Flanagan and Patricia Korta. Their unwavering support during my entire academic career has helped me overcome the challenges that presented themselves along the way. Their willingness to push me, even when I did not particularly like them for it, guided me to become the person I am today. Also to my brother, James, for providing me with something to strive for, and to my sister, Katie, for keeping me grounded.

Acknowledgements

First, I'd like to acknowledge Dr. Erich Schneider for his guidance and support throughout this entire process. His knowledge in this field was invaluable when preforming this work.

I'd be remiss if I did not mention the support that Dr. Anthony Scopatz has given throughout the past five years. The mentoring he provided during my early years as a graduate student on proper coding techniques was invaluable when developing the software that supported the methods discussed in this work.

Cem Bagdatlioglu for his assistance with developing Bright-lite. His willingness to learn and dedication to improving the software made working with him and on the software a delight.

I'd also like to acknowledge Dr. Yarden Livnat for his guidance and suggestions while developing the visualization software discussed in this work. In addition, I must acknowledge the Cyclus team (specifically Dr. Paul Wilson, Dr. Katy Huff, Dr. Matthew Gidden, and Robert Carlson) for their technical support on using and interacting with the Cyclus software.

Finally, Dr. James Armstrong, Dr. Kenneth Dayman, Birdy Phathanapirom, William Gurecky, and the members of NR for their friendship and support on this journey.

Novel Methods for Generalizing Nuclear Fuel Cycle Design, and Fuel Burnup Modeling

Robert Ryan Flanagan, PhD

The University of Texas at Austin, 2015

Supervisor: Erich Schneider

The large number of reactor designs and concepts in existence open up a vast array of nuclear fuel cycle strategies. u. These different reactor types require unique supporting systems from raw material extraction and handling to waste management. Any system designed to model nuclear energy should therefore have methods that are capability of representing a large number of unique fuel cycles. This work examines a user interface designed to generalize the design of nuclear fuel cycles. This software, known as CycIC, allows users to interact graphically with a fuel cycle simulator (Cyclus). In this work, the capabilities of CycIC were improved through two rounds of rigorous user experience testing. These tests were used as a basis for implementing improvements to the software. Two views inside the software were improved to allow for users to interact with the software more intuitively, and features that provide help to the users were added to improve understanding of fuel cycles and Cyclus. Additionally, this work expands the capabilities of a reactor modeling software (known as Bright-lite) which uses the fluence based neutron balance approach to determine burnup, criticality, and transmutation matrixes for nuclear reactors to augment its modeling of the broadest range of fuel cycle strategies. Specifically, a multi-dimensional interpolation method was implemented to enable reactors to be characterized by sets of cross section libraries which potentially depend on a large number of reactor characteristics. The accuracy of this interpolation method is demonstrated for a number of

parameters for light water reactors, and techniques for using this interpolation method to automatically generate reactor libraries for Bright-lite are demonstrated. This research also generalizes the ability of the Bright-lite to blend multiple streams of nuclear fuel while still maintaining constraints. This system is demonstrated for continuous recycle nuclear fuel cycles utilizing light water and fast spectrum reactors. The results show that Bright-lite is capable of blending fuel to reach several targets using up to three different input streams.

Table of Contents

List of Tables

List of Figures

INTRODUCTION

The future of energy generation in the nation is a political and social issue that is on the forefront of nation's evolving political landscape. To this end, it is important to make sure that both the nation's population and those with political power understand how nuclear energy can fit into the mix of energy generating technologies. Accomplishing this requires these groups of people to have access to modelling software that can demonstrate and quantify the advantages and disadvantages of different reactor technologies and nuclear fuel cycles. Additionally, it is important that everyone have easy access to an intuitive and informative method of interacting with the nuclear fuel cycle. This requires a clean and concise input interface, and a tool for quantifying and comparing the output metrics of a fuel cycle.

A new fuel cycle simulator (Cyclus) developed at the University of Wisconsin at Madison is capable of interacting with independently developed modules that allow it to simulate a wide variety of technologies and political behaviors. This scope gives rise to a large set of possible future nuclear fuel cycles. Analysis of the full set of technologies and nuclear energy strategies will require the capability to quickly generate and simulate nuclear fuel cycles scenarios.

To develop such a large set of nuclear fuel cycles requires a method of modeling each of the reactor technologies that are currently available or may become available in the future. Each of these technologies can be parameterized in a number of different ways, and therefore modules representing them must be flexible in their behavior.

In advanced fuel cycles the isotopic composition of input and output fuels may change over time. This occurs because the fuel being put into a reactor during start up does not match fuel the reactor uses during steady state operation (several cycles after

startup). Reactors may undergo other transients as well. A power uprate (increase in the thermal power output of the reactor) will require the reactor to burn more highly enriched input fuel. Other reactors might start burning one type of fuel but switch over to a new fuel after several cycles. Finally, any reactor that uses the spent fuel from a reactor undergoing any of these changes will also be subject to a changing fuel composition as the spent fuel will also change with the input fuel. Therefore the ability of a reactor module to accept a dynamic input fuel composition is of high importance.

To accomplish these goals a new reactor burnup simulation module being developed for the Cyclus fuel cycle simulator uses prebuilt burnup and output composition libraries to calculate the input and output isotopic composition. The method used by this module is fast and computationally inexpensive. Therefore a large number of reactor technologies can be investigated quickly.

The libraries used in this module are time consuming to generate; therefore producing the libraries to represent all possible reactors is not feasible. As part of this work a method for library interpolation is being developed to provide increased fidelity by dynamically creating libraries interpolated from existing libraries. The interpolation method is based on an inverse distance interpolation method. This method allows the interpolation scheme to operate effectively in non-uniform grids over many different variables; burnup, fuel composition, fuel pitch size, etc.

Another research challenge is developing a system for blending multiple input fuel streams. A reactor in an advanced fuel cycle may be taking spent fuel from one or more upstream reactors. The spent fuel from upstream reactors will differ, and even the spent fuel from a single reactor might even be different at different times in a fuel cycle.

A good example of this would be a closed fuel cycle involving a fast reactor. The fast reactor starts up using a combination of light water reactor spent fuel and depleted

uranium. Eventually the reactor will start accepting its own spent fuel, which has a different isotopic composition than light water reactor spent fuel.

To determine the best blending of these streams the module will combine streams in an attempt to meet or maximize constraints (burnup and multiplication factor, minimizing an isotope, fluence and neutron-induced damage per atom, etc.) the user has put on the system.

It is important to allow users to apply these constraints, and indeed to interact with nuclear fuel cycle simulators in a flexible and extensible manner as the simulators themselves evolve. The majority of the user base of nuclear fuel cycle simulators consists of people experienced in the nuclear engineering field. Persons outside of the nuclear engineering community lack the understanding common terms and situations within nuclear engineering which would allow them to interact in a meaningful way with current simulators. Therefore a more intuitive method of interacting with nuclear fuel cycles is required if the user base of this type of software is to expand to the general public and public policy makers. The challenge faced in this work is how to know if the user interface is being effective at opening up fuel cycle simulation to the general public.

A user interface is being developed to couple with the Cyclus fuel cycle simulator. This user interface, Cyclist, is composed of both front end (CycIC), and back end (Cyclist), visualization software. These systems incorporate many features and lessons learned from successful graphical user interfaces (GUIs) that allow them to be user friendly and intuitive.

CycIC (Cyclus Input Control) is software developed in Java that provides features to enhance the user experience with the simulator through views for visualizing flows between technologies and regions. Additionally, CycIC contains form generation software that will read the inputs required for any Cyclus module. From these inputs

CycIC generates a descriptive and easy to read form that the user can fill out. These forms reduce the burden of learning common nuclear engineering terms and make it much easier for non-nuclear engineers to create fuel cycles.

Testing how smoothly users can interact with the CycIC software is achieved through “user experience” visualization experiments. These experiments are focused on getting actionable, qualitative and quantitative feedback on how quickly users learned the software, how much difficulty they had accomplishing certain tasks, and how the tool can be improved. During the experiments user interaction with the software will be monitored to determine where weaknesses might exist. Additionally after each experiment users will be asked questions about specific components of the software.

Coupling a powerful user interface with a robust and flexible new fuel cycle simulator with modules capable of quickly and accurately performing calculations on a large number of possible reactor combinations will allow both the general public and technical and nontechnical decision makers to better understand nuclear technologies and make the appropriate choices for the future of the nuclear industry.

LITERATURE REVIEW

2.1 Visualization of User Input to Fuel Cycle Simulators

User interaction with fuel cycle simulators has been limited to text based inputs or graphical systems designed as an afterthought for the simulator. These systems lack features which clarify difficult to understand input fields or require multiple programs to generate simulations.

2.1.1 ORION

Some fuel cycle simulators such as ORION [1] are coupled with more recently developed graphical user interfaces. ORION is a fleet based simulator. In a fleet based simulator facilities of the same type are lumped into a single fleet. ORION is stuck behind a paywall (the software costs money to own) and therefore not likely to be used by the general populace. The paywall sets a monetary barrier to entry for users outside of the nuclear engineering field. Therefore their aim is not to provide a user interface for the general public but for persons in the nuclear engineering field. CycIC is aimed at providing a user interface for people outside of the nuclear engineering field as well as those inside.

Part of accomplishing this means that CycIC must be open source software. This provides the benefit of making it available to the general public for free. However, comparisons between software like ORION and CycIC are difficult because they have two different target audiences.

Figure 1 [2] provides an example of how a fuel cycle is visualized using the ORION interface. This visualization allows the user to see how materials flow from one facility to another. To fully describe a fuel cycle using the ORION software requires a

great deal of facility types. For this fuel cycle there are 60 objects and 85 flow pathways. Other fuel cycles may require even more objects and flow pathways.

Given this number of objects, and the requirement that even more might be present on the screen, facility types are color coding to aid in identification and arrows show flow pathways between facility types. This allows users to know the type of facility at a glance.

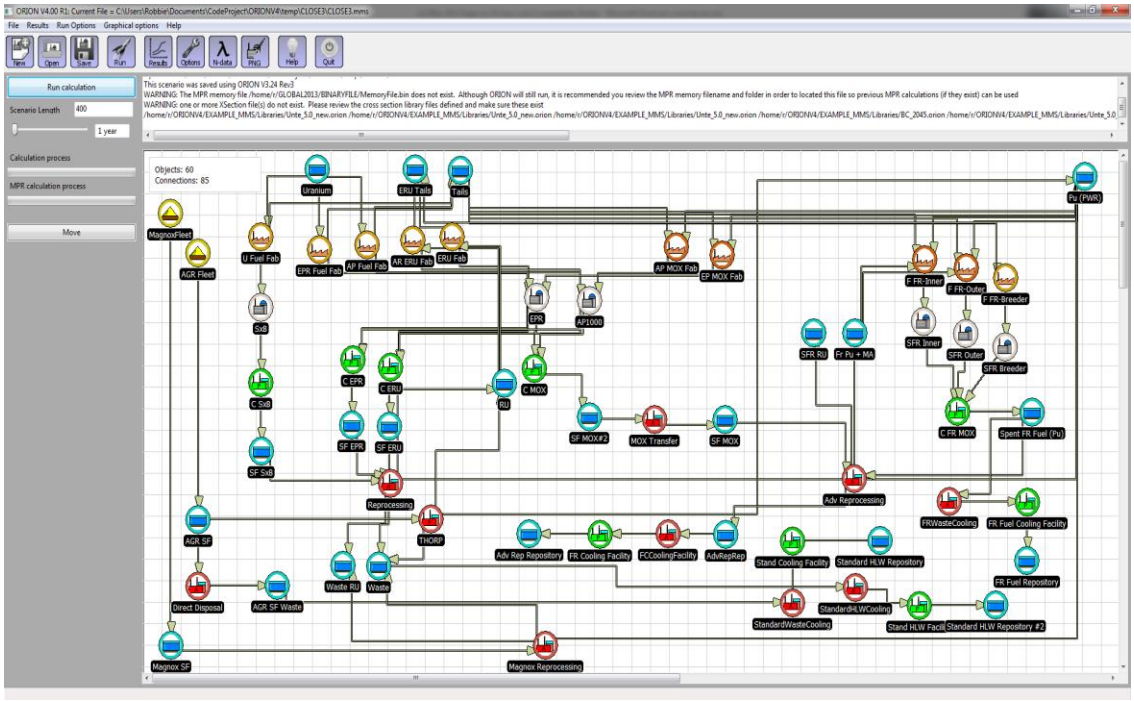

Figure 1 A visualization of ORION's user interface system with a closed nuclear fuel cycle.

The control panel on the left hand side allows users to manipulate some options of the fuel cycle, however to modify facility information and facility types users must change that in the options tab (or inside the code).

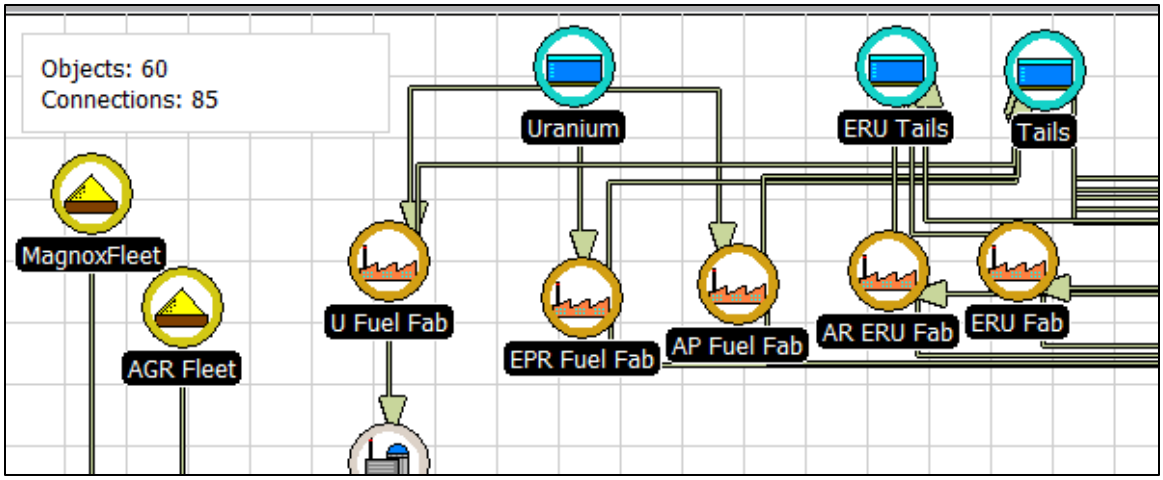

Figure 2 Close up of the facility type symbols used by ORION and the naming structure.

Figure 2 provides a close up of several of the objects from Figure 1. Names are applied to each object below the object for identification. Additionally, this close up view shows the graphics associated some with the fuel cycle types. These graphics are also useful for providing the user with an idea of what type of facility the object is. The flow directional arrows can also be seen more clearly in Figure 2.

2.1.2 VISION

While no simulator has had the goal of being supported by a user interface designed for the general public, one simulator, VISION, is widely used in the United States today by the nuclear engineering community. This simulator will provide a point of reference for comparison to CycIC's capabilities.

The user interface of VISION is powered by the software PowerSim used in conjunction with Microsoft Excel [3]. PowerSim is used to set up diagrams for the flow pathways between facilities through the use of flows and levels [4]. Flows represent path ways in which materials can move. Levels represent inventories of materials. These levels represent the inventories of different facilities within the fuel cycle. PowerSim

supports two other main objects as well; constants and auxiliaries. Constants are typically inputs to the simulation and auxiliaries are equations that depend on time, constants, or other auxiliaries. The visualization of each of these objects is as follows.

- Flows – Directional lines ending in arrows
- Levels – Rectangles
- Constants – Diamonds
- Auxiliaries – Circles

These structures are coupled together to create the mass flow rates between facilities in a nuclear fuel cycle. An example of this can be seen in figure 3.

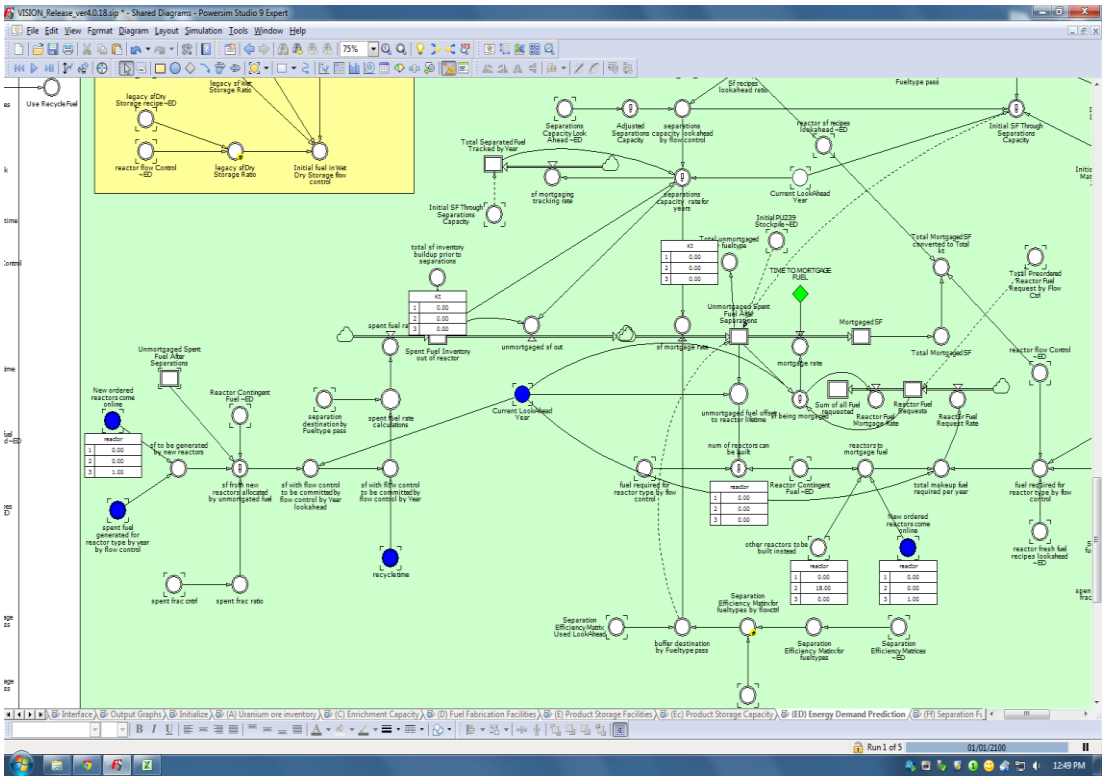

Figure 3 Visualization for the reactor systems inside of VISION.

As more facilities and operations are added to the system, screen clutter may become a problem. Reducing on screen clutter has proven to be a very successful technique in terms of user satisfaction with graphical user interfaces. [5]

In most cases users will not be interacting with this view, VISION comes with prebuilt fuel cycles. However, if a user needs to modify the fuel cycle in any way they must directly interact with the PowerSim interface. To do so they must find the exact component they wish to interact with and modify it.

Figure 3 also indicates that there are several more tabs in this PowerSim visualization. Each of which corresponds to more features of the VISION software or different parts of the nuclear fuel cycle. Many of these tabs look similar to Figure 3. This separation of components of the fuel cycle to different tabs means there is no way to visualize the full nuclear fuel cycle in one screen.

VISION also uses Microsoft Excel spread sheets to store the recipes, facility properties and simulations constants. Figure 4 shows an example of a spreadsheet used in the VISION system.

There exist several spreadsheets that must be filled out to be used with VISION simulations. Again, all of these spreadsheets come prepackaged with VISION. If a user wanted to modify a facility recipe or constants inside of the simulation however, it would require that user to go into the spreadsheets and modify the values in question.

In addition to the clutter, it has been shown that users prefer to see direct interaction between objects when working with a user interface. If a user wants to modify parts of an object the preferred method is through directly interacting with that object first and then modifying it [5]. An EXCEL spreadsheet is not an effective method of showing this relationship.

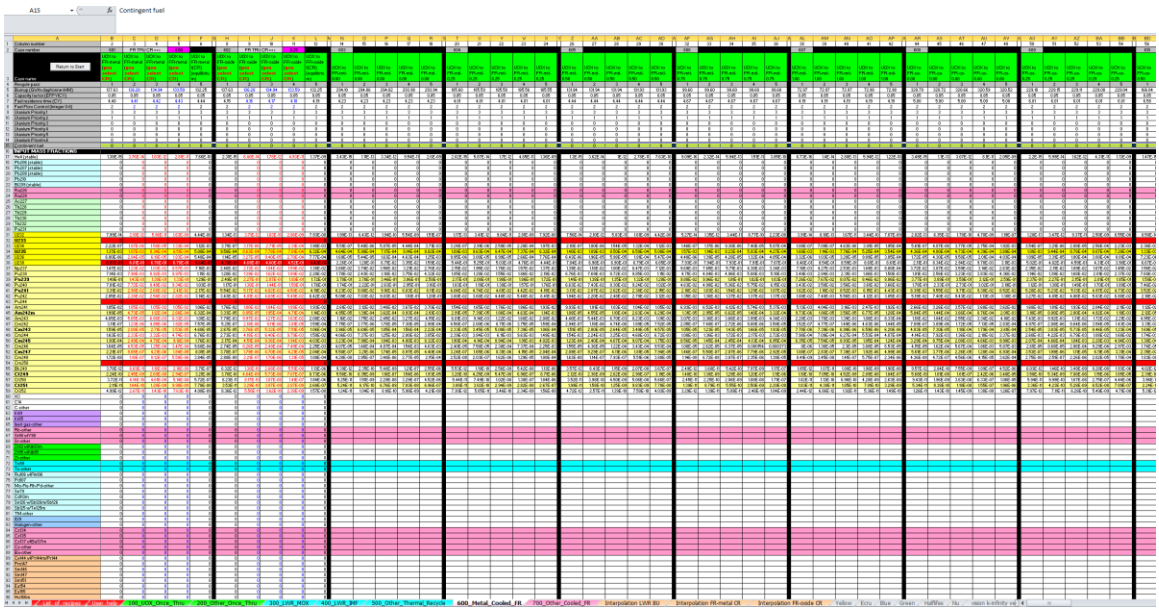

Figure 4 Spreadsheet for a fast reactor facility in the VISION recipes spreadsheet.

2.2 VISUALIZATION EXPERIMENTS

A focus of this work is to create experiments to test the visualization approaches taken by the CycIC system. There are two goals of these experiments. The first is to drive the future development of features to CycIC. The second is to determine which areas of the CycIC user interface give users the most difficulty in terms of the following attributes; time to learn, visuals, and functionality. This information can then be used to drive further visualization experiments to determine a possible solution to these problems.

One method for testing the CycIC visualization is qualitative experiments that track how a user interacts with the system over the course of the experiment. Techniques for doing this including asking users to speak their thoughts out loud as they are going through the experiment, or to prompt them on with questions as they work on certain parts. Experimenters may also which to monitor how often they interact with a specific portion of the user interface or how much time they spend trying to achieve their goal

using a certain view. Finally users can be asked to fill out qualitative questionnaires on their experience with the software [6]. These types of visualization experiments are often called Human Computer Interaction experiments [7]. These experiments incorporate case studies in which experimenters watch and interact with subjects as the subjects interact with the software being tested.

One such visualization experiment conducted on the Hierarchical Clustering Explorer (HCE) heavily used participatory observations (observations made by the experimenters) and interviews with the subjects [8].

Subjects met with researches every four to six weeks to use HCE for roughly 30 minutes. At the first meeting, users were instructed on how to use HCE and given examples of its use. At all other meetings the users were asked to accomplish tasks with HCE or attempt to accomplish their own tasks. While this was happening researchers recorded their impressions, implementation requests, and asked a series of questions to examine their experience with HCE.

These questions were focused on the following aspects.

- How does HCE improve the way users analyze multivariate data sets?
- How does the score overview help users identify interesting projections?
- How does the histogram browser help users traverse projections?
- What are the most frequently used ranking criteria?
- Identify possible improvements in HCE.

Additionally during the period between meetings, the researchers attempted to develop the implementation requests for the subjects. This way subjects could provide feedback on these updates.

2.2 Fuel Composition Generation and Library Interpolation Methods in Nuclear Fuel Cycle Simulators

Fuel cycle simulators track the movement of materials through the different facilities within a fuel cycle. In the case of nuclear reactors, the reactor will burn the material to produce energy. Reactors can burn fuel composed of several different isotopes, to determine the exact composition of this fuel the reactors in fuel cycle simulators use a variety of methods. Additionally reactors use interpolation methods to reduce the amount of computation work associated with calculating these compositions.

2.2.1 VISION

The Verifiable Fuel Cycle Simulation (VISION) code has been developed as a PowerSim application by the Idaho National Lab (INL) [3]. To develop the reactor models used in VISION, external neutronics codes – MCNP for example - are run to generate information on the burnup, criticality, and output isotopes of the reactor.

Reactors in VISION require pre-calculated recipes for both their fresh and spent fuels. The input recipe is the fuel feed into the neutronics code, and the output recipe is the isotopes in the fuel at discharge. The recipes only correspond to specific reactor types and fuels for those reactor types.

In a nuclear fuel cycle, fuels can be recycled and therefore the isotopic compositions of fuels are subject to change. VISION correlates between this mismatch of input/output recipe isotopics using a single independent variable. Additionally this interpolant is specific to the reactor and fuel type. For example, in the LWR UOX type fuel the correlation is structured as a function of reactor burnup between 33 and 100 MWth-day/kg-iHM. For a fast reactor the correlation variable is the conversion ratio of the reactor, ranging from 0 to 1.0.

This limits the ability to interpolate between reactor libraries that have only one difference between them, whatever the correlation for that reactor type is. Therefore if a user wanted to interpolate between reactor libraries of a different correlation factor, for example enrichment, a new reactor type would need to be generated for VISION with recipe libraries at varying enrichments.

This correlation is method allows VISION is solve an issue often referenced to as the "winery problem" [9] by users and developers of VISION. The problem, briefly mentioned before, is that the composition of spent fuel to be reprocessed may change during a reactors lifetime. This creates an issue because the input composition of fuel for a reactor is typically held relatively constant for a simulation. Therefore a mismatch between fuel to be reprocessed, and target fuel to be produced through reprocessing is created.

2.2.2 NFCSIM

The Nuclear Fuel Cycle SIMulator (NFCSim) models the movement of materials through the nuclear fuel cycle with a time dependent event driven system [10]. The simulator is written in the computer language Java, which uses classes to represent objects and functions.

To perform it the burnup calculations required for simulating reactors in the nuclear fuel cycle NFCSim uses a criticality and burnup engine. Like VISION, NFCSIM uses reactor-specific models within its burnup engine. The engine computes the reactivity of the core using a piece-wise linear reactivity model of the core. Each reactor model in NFCSim is represented by a set of one group cross sections.

Due to the software's reliance on one group cross sections the addition of new reactor types, or existing reactor types with new parameters is predicated on the

development of these one group cross sections. These one group cross sections must be generated externally to NFCSim which puts a burden on the user.

To perform the burnup calculation during runtime NFCSim must make an external call to ORIGEN2. The one group cross section libraries are used to generate the burnup and neutron production/destruction rates of each actinide present in the initial fuel. The initial composition of incoming fuel is determined by creating a linear combination of the actinides available to reactor that ensures the reactor reaches the desired burnup by the time the core becomes subcritical.

NFCSim has methods for blending nuclear fuel streams for three types of reactors. Each reactor has a specific algorithm designed to blend the fuel to match a target output burnup. The three reactor types supported are mixed oxide reactors, accelerator driven systems, and fast reactors [11].

2.2.3 CAFCA

The Code for Advanced Fuel Cycles Assessment (CAFCA) is a fuel cycle simulator developed in VENSIM by Massachusetts Institute of Technology [12]. Due to assumptions made about the nuclear fuel cycle embedded into CAFCA, it is assumed that all non-reactor facilities will increase their production rates to match the need of facilities down the line. The limiting factor for the fuel cycle then becomes the availability of spent fuel to be reprocessed. Recycled fuel can take two forms, either U-TRU fuel or MOX fuel, and does not track the individual Pu, reprocessed uranium or uranium-transuranic inventories. CAFCA requires the user to input a desired ratio for these two fuel types, which the software will then try to match by making decision about the deployment of the fuel types.

An additional assumption made in CAFCA is that fuel compositions are specified at the elemental, rather than the isotopic level. This requires that the input and discharged fuels be treated as if they are at an equilibrium situation for the entirety of the fuel cycle. This assumption prevents the modelling of reactor transients, that is start up and shut down of a reactor. Additionally the availability of only two paths for reprocessed fuel is very limiting in terms of modeling new methods of burning spent reactor fuel. CAFCA is currently capable of modelling only a closed fuel cycle using light water and fast reactors. It is incapable of blending fuel streams to produce a mixed fuel stream.

2.2.4 COSI

Commelini-Sicart (COSI) is a nuclear fuel cycle simulator written in Java, developed by the CEA, the French Atomic and Alternative Energies Commission. COSI also gives the users the ability to choose how spent fuel will be burned in the simulation, by allowing them to set a preference based on the age of the fuel.

Given that the isotopic vector of the input fuel is known (the isotopes in input fuel are assigned by users), COSI determines the isotopic composition of the fuel using an equivalence method which takes a reactor and fuel type specific approach. It does so by blending together steams of fuel input. There are two classes, or types, of material. The first class is fertile material, and the second class is fissile material. The input composition is therefore a ratio of the two classes. For a UOX reactor the fuel used is modeled by setting an enrichment of the fuel (that is a blending of Uranium 238 and Uranium 235 for each class respectively). MOX fuel is modeled by determining the required plutonium (a given stream of the output plutonium from a reactor) content of the fuel. Fast reactors are modeled in COSI by determining an ideal Pu-239 and U-238 loading of the reactor, and adjusting the input for the deviations from this ideal.

One major gap in COSI is the limitation to two streams of material. This gap is made very apparent when considering a fast reactor. A fast reactor might have an input stream of fuel from a light water reactor, an input stream of fuel from a fast reactor, and it may have top up fuel in the form of depleted uranium. The capability to accept more than two streams of fuel is very important to creating a fuel cycle simulator that is fully capable of describing a nuclear fuel cycle.

CYCLUS INPUT CONTROL USER EXPERIENCE TESTING

3.1 CycIC

One of the main goals of the Cyclus project is to open investigation of nuclear fuel cycles up to more than just specialists in the engineering community. This will make the long term benefits and consequences of technologies and fuel cycle choices more accessible to the general public.

CycIC is designed to replace the text based input method that is the standard method of input for Cyclus. The graphical user interface (GUI) will be designed to allow even inexperienced users to explore nuclear energy options through the simulator. CycIC accomplishes this through the use of a series of views that allow the user to directly interface with their simulation.

Cyclus is capable of accepting user developed modules into its framework. A module is a piece of code developed to represent a facility within the nuclear fuel cycle. Modules may also represent regions and institutions. CycIC must also be able to support this capability. In order to accomplish this, CycIC needs to be able to discover and read the input structure of all Cyclus modules on a local or remote machine.

Due to the flexibility of Cyclus, it is important to design a user interface that is easy to use yet remains capable of fully leveraging the abilities of Cyclus. This means that users must be capable of constructing an arbitrary fuel cycle quickly. The CycIC tool hopes to achieve the following major goals;

- Couple with Cyclus to determine which models are available on the current system.

- Provide users with a visual representation of the nuclear fuel cycle.
- Provide users with a visual method of interacting with facilities within the nuclear fuel cycle.
- Give users assistance in understanding key components of the nuclear fuel cycle through tooltips, prompts, and help menus.
- Generate working Cyclus input files

3.1.1 Cycle View

The Cycle View provides the user with a method of visualizing the fuel cycle they are creating through facility nodes (circles that represent facilities) and flows (arrows to represent material flow directions). The Cycle view is seen in Figure 5.

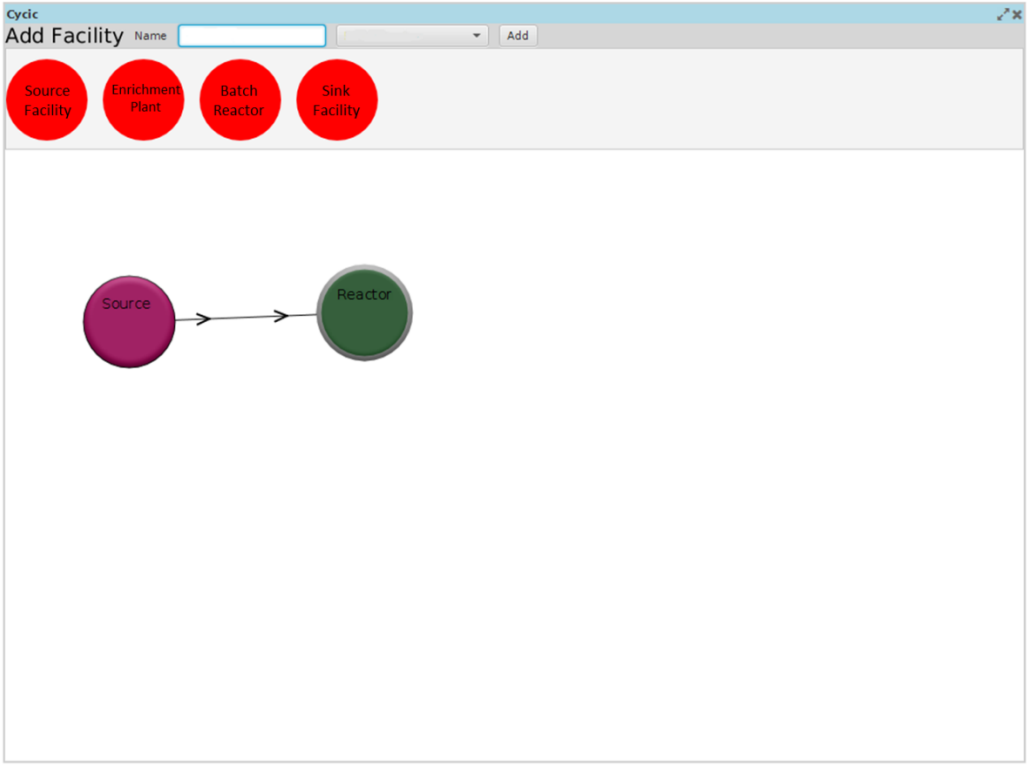

Figure 5 Cycle View (Main View)

There are two main components to this view. The first is a scrolling pane that contains the facility types that CycIC has located on either the local or remote machine. From this scroll pane users can drag and drop the module nodes and place them on the second component of the Cycle View. Here the user will be able to visualize the fuel cycle. Nodes dropped onto the flow control view can be moved around and repositioned, allowing users to visualize their fuel cycle how they like.

3.1.2 Region and Institution Corrals

These views provide the user with views and forms for interacting with the regions and institutions required for a Cyclus simulation. Regions provide the user with the ability to provide conditions on the future of the simulation. An example of this would be specifying the growth profiles for the demand of nuclear energy. Institutions allow users to set constraints and build orders that define when facilities are allowed to be constructed. Facilities must be attached to institutions, which in turn must be attached to regions.

The region corral view can be seen in Figure 6, the region form view is visible in Figure 7, and the institution view can be seen in Figure 8.

The focus of the region corral is to provide the user with a mode of creating and controlling regions. Regions are represented by nodes (region nodes are squares) on a view pane. These nodes can show the movement of materials between regions; by investigating the facilities within each region, the material flows between facilities can be utilized to show the material between regions if any connected facilities exist in different regions. Region nodes also allow the user to open a form that allows the user to set information about the region. The institution view opens a form that allows the user to fill out information about the institution.

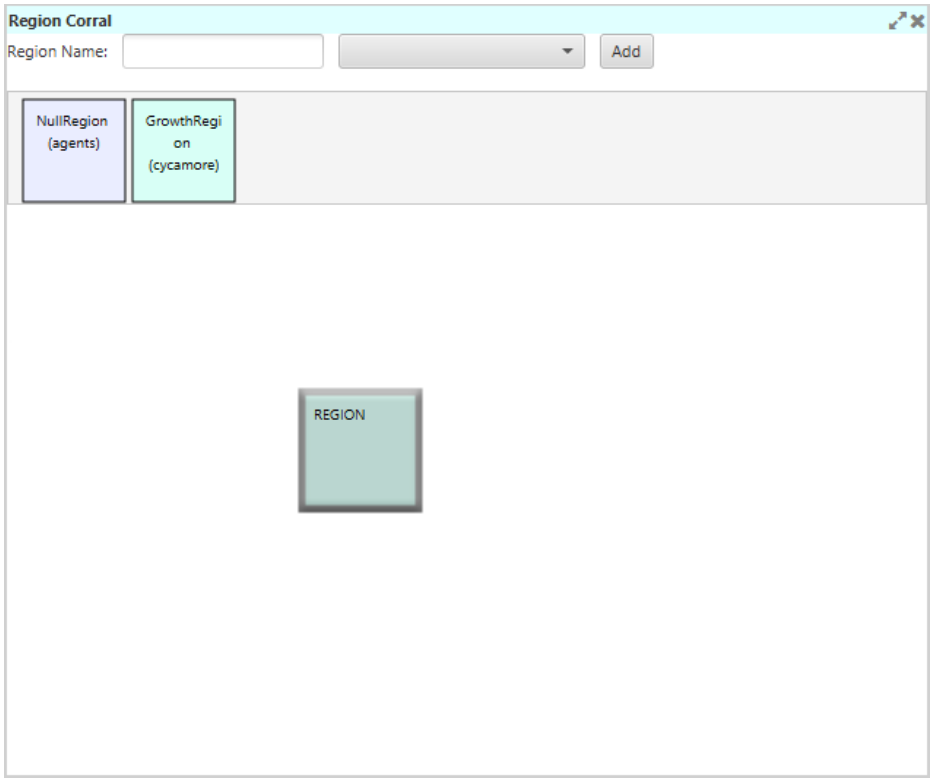

Figure 6 Region Corral View

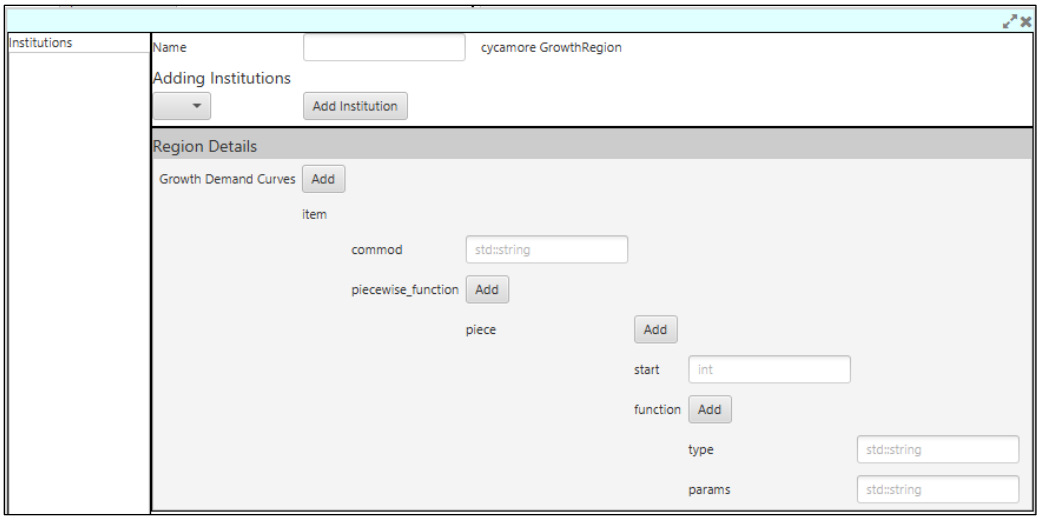

Figure 7 Region Form

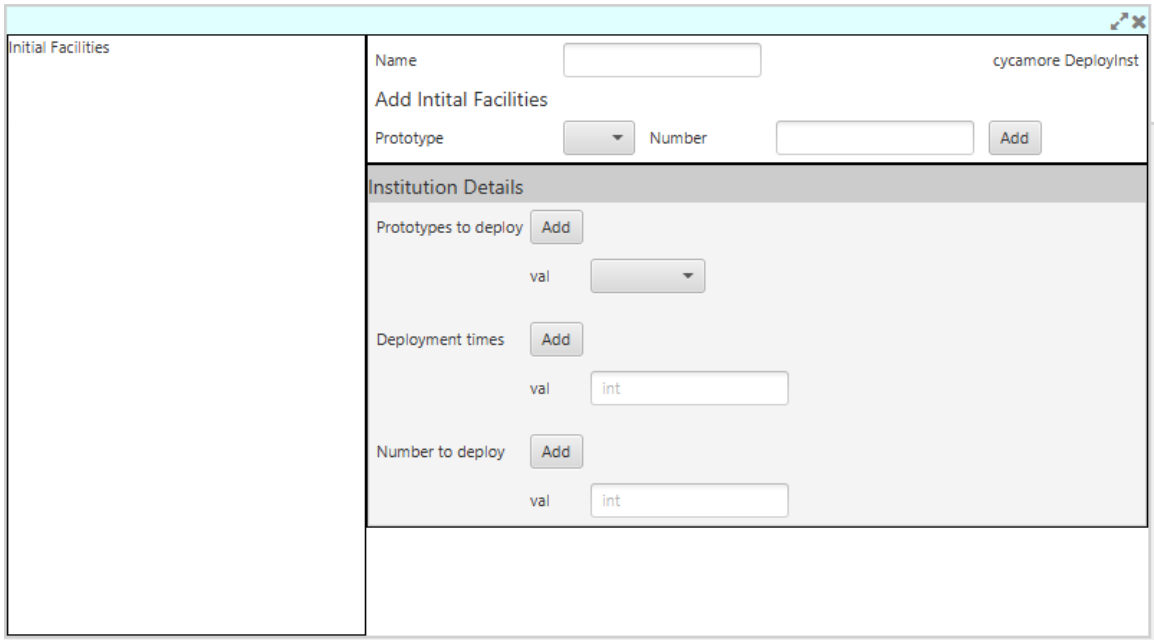

Figure 8 Institution View

3.1.3 SIMULATIONS DETAILS

This view allows the user to directly control the top level requirements for a Cyclus simulation. This is where the temporal information for the simulation is set; the start date, and the duration of the simulation. Users also create the commodities within a simulation through this view. Figure 9 shows the simulation details.

Additionally this is where users can write several notes on the details of their simulation. These notes can be used to indicate conditions in the user has put into the simulation, or any other comments than the user might want to have associated with their simulation.

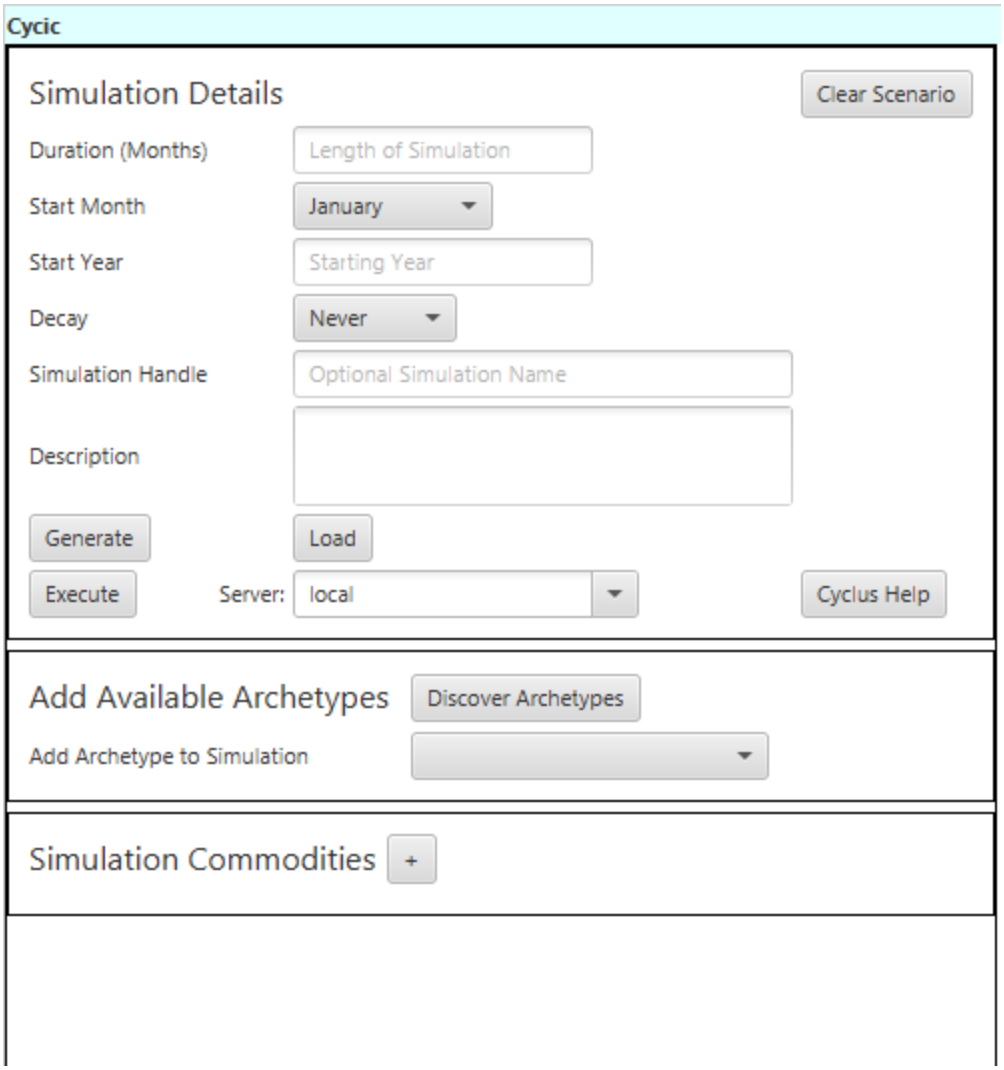

Figure 9 Simulations details

3.1.4 Recipe View

The recipe view allows the user to create the recipes that they require within the simulation. A recipe is a collection of isotopes and the amount of mass (or atoms) associated with that isotope in the collection. Within this view the recipe type may be set: either on a mass or atom basis. The user may then set the isotopic content of the recipes and their corresponding masses/atom fractions. Figure 10shows the recipe view.

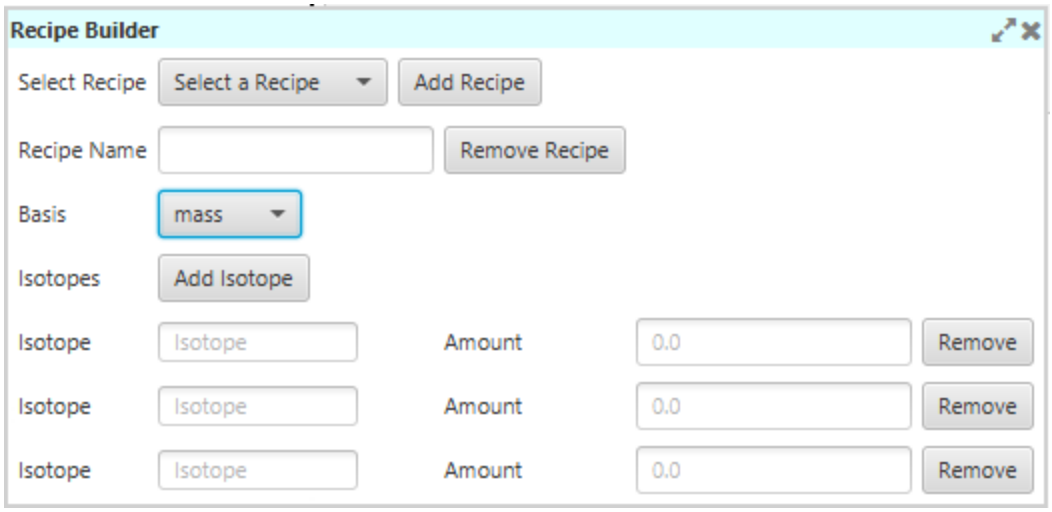

Figure 10 Recipe Generator View

3.1.5 Form Builder Functions

The flexibility of Cyclus presents a challenge to the user interface. Cyclus modules will contain inputs that the user will need to fill out in order for the module to properly interface with Cyclus. An example of this may be the burn-up of a reactor.

It is also likely that each facility, region, or institution module will contain many of these input fields that the user is required to enter. CycIC's form generation methods take all of these input fields and use them to generate forms that the user can interact with inside the user interface. Figure 11 depicts an example of a form generated by CycIC.

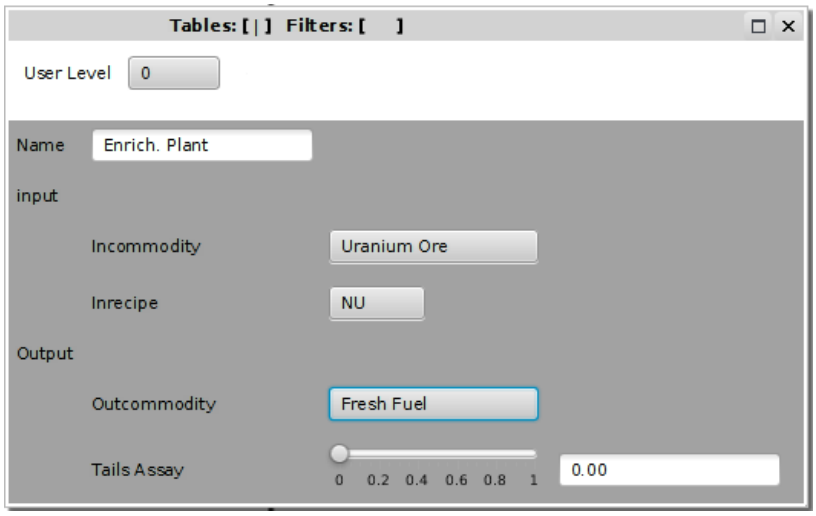

Figure 11 Facility form for Cyclus Enrichment plant

CYCIC allows developers and users to set the "user level". This will set defaults for some features and hide some features for less experienced users, allowing them to unlock more advanced input fields if they desire. Default values are required for all input fields for a facility (set by the module developer) that have a user level greater than the lowest level. If a user sets the user level of the form to below that of the user level of a input field that field will disappear from the form and the default value will be taken as the value for the field.

This feature is important for opening Cyclus up to the non-technical user. While tooltips and help fields are helpful in describing features, beginner users may want to hide more advanced fields within a form to reduce clutter that might overwhelm an inexperienced user.

Using reactor burn-up as an example, if the reactor burn-up input field has a user level of 1, then when the user sets the user level to a value less than 1, the burn-up field in the form will disappear, and the default burn-up value will be used.

Figure 12 shows how the user might set the user level for a specific form and how the forms change due to this change.

One issue that arises from allowing developers to create their own modules is that input fields to a module may have names that are confusing. An example of this might be an input field known as “corereload”. The additional difficulty of having coding constraints (no spaces in coding constraints) means that names may not be fully explanatory of what they refer to.

To reduce this problem developers have been given the ability to add tooltips to their variable fields. This will allow users to mouse over a variable name on the form to reveal a short sentence or description of what the variable name or field means (Figure 13).

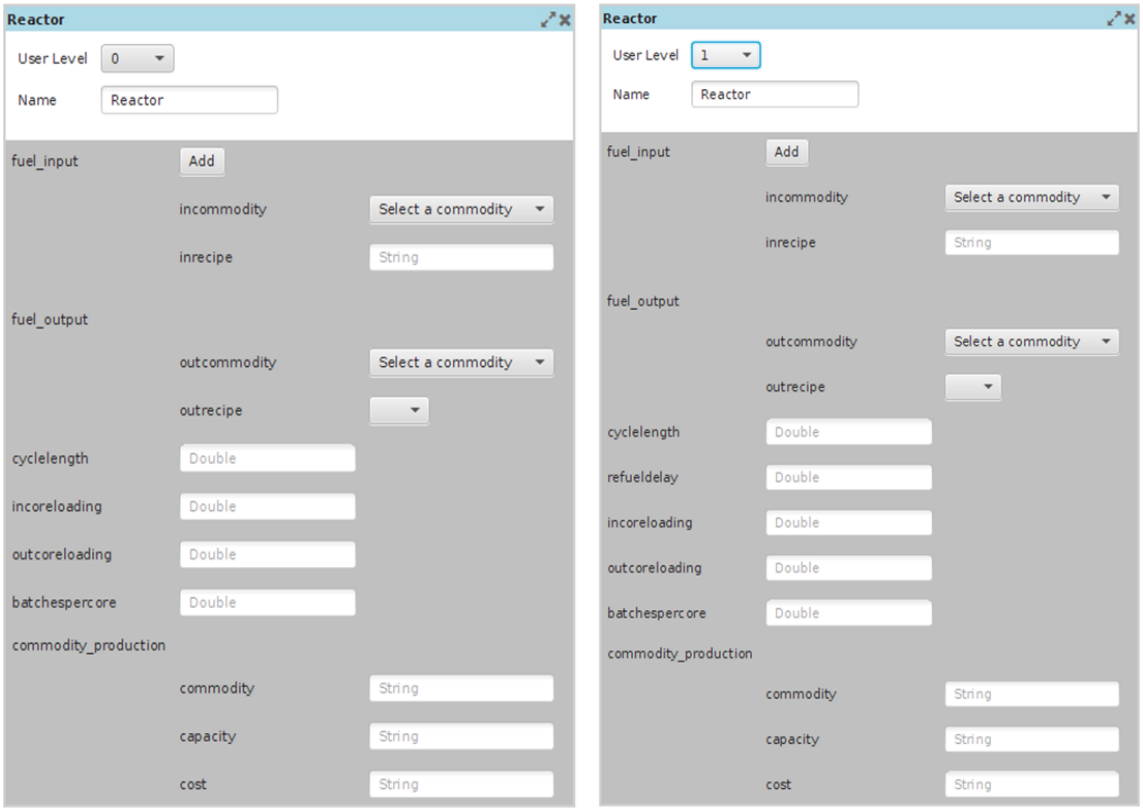

Figure 12 User level comparison for batch reactor. Addition of refuel delay field.

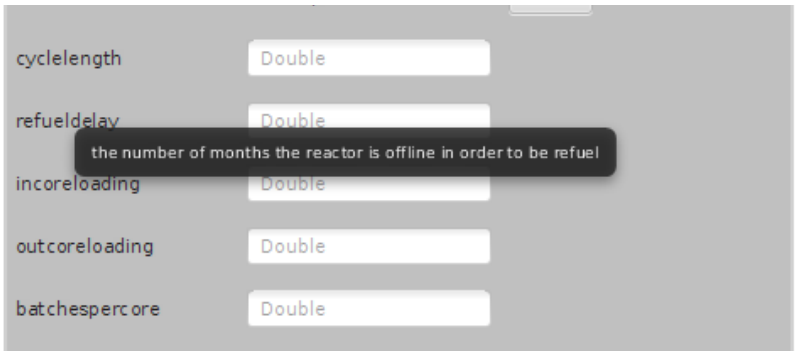

Figure 13 Tooltip demonstration

Additionally CycIC has built in functions apply special methods for the user to interact with a form. For instance a developer might want to limit the value of a specific field to a certain range. In this case the developer can apply a tag to their variable that will change the field's input style from a standard text field to a range slider. An example of this can be seen in Figure 11. In addition to range sliders the developer has the ability to add other specialized fields including drop down menus.

Developers may wish to include specifically designed form structures or visualization tools to their modules. While it is often difficult to code for unknown use cases in the future, because CycIC is open source software developers will also be able to build modules for CycIC that might do a better job of representing input fields in their facilities than the methods currently available in CycIC.

3.2 Visualization Experiments

User experience testing is a type of visualization experiment that has users sit down and interact with computer software. During this interaction the users are either given a task to complete to or instructed to use the software in a way that they would normal use

it [1][2][3]. This testing can be conducted at any point in software's lifetime and thus the users can have varying levels of experience with the software. However given that the CycIC software is still currently in development the users in this experience testing will all be new or relatively new users to the software.

3.2.1 User Experience Testing Round 1

3.2.1a Tutorial

The first round of user experience testing on the CycIC software was conducted using engineers with various different areas of expertise. As these users were brand new to both CycIC and Cyclus a user tutorial was provided for the users before the testing began. This tutorial was performed for a majority of the users twice; once in a large group setting, and a second time directly preceding the user experience testing for each of the users. The purpose of this tutorial was to make sure the users had a good understanding of the basics of the software and understand the reasons behind some of the actions that they were asked to perform.

The tutorial consisted of a brief introduction to the concepts and ideas behind the Cyclus fuel cycle simulator. This included a description of the major components inside the Cyclus software; archetypes, prototypes, agents, regions, institutions, commodities, and recipes. Additionally the users were instructed on how each of these pieces was connected. To aid in the understanding of how these components were featured in the CycIC software while these descriptions were occurring the person facilitating the user experience testing (the observer) was demonstrating how each of the Cyclus components is represented inside CycIC.

3.2.1b User Experience Testing

The second portion of the testing started when the users began to use the software. Users were given a task to complete using the software and their actions were tracked by an observer. In this test the task was to construct a simple fuel cycle consisting of three prototypes and add key components to the Cyclus simulation.

Before users began this portion of the testing, they were asked to maintain a running self-narration of their actions and thoughts during the process. This is a common method used in this sort of testing for two reasons. First, this method provides the observer with information that might otherwise not come up in a questions and comments section after the testing is completed. Thus, this method is useful in gaining additional information from users about the software [1] [3], as is possible for users to forget issues that arose during testing if other issues come up later. The second benefit this method has is to allow the user to complete the task without their actions being interrupted by questions asked during the testing. This prevents distractions from causing the user to lose their place and might confuse the user while they are operating the software [1] [4]. During the test the observer was seated next to the user and actively listened to their comments and recorded these comments as best they could.

However, the main job of the observer during this portion of the testing was to recording the amount of time (and number of clicks) it took for the users to accomplish certain tasks and how long they spent using certain views. These two values were recorded because they represent indicators of how smoothly the user experience flows.

The purpose of this graphical user interface is to reduce the barrier of entry of Cyclus to a user, and to provide a quick and easy way of generating input files. Therefore the amount of time it took to complete various tasks was considered because it gives valuable feedback on if the graphical user interface is providing a quick method of generating an input file. If users spend more time using a view to complete a specific task than they would by generating the input file by hand, it may indicate that that portion of the software should be updated or streamlined.

The number of clicks was considered because these represent direct interaction between the user and the software. The greater the number of click, the greater the effort the user is putting into the software. Therefore this is used as an indicator of how much work is being done by the user, and provides the software developers with information about where the users spend the most effort. This information can be used by the CycIC developers to determine where their efforts should be focused.

Additionally the time and number of clicks was considered to compare two views. The institution view and the region corral. These two views provide very similar functionality, but represent two different methods of input. The institution view is primarily a text based input (a large form) and the region corral is a more visual experience (moveable nodes and separate forms).

To limit the amount of nuclear fuel cycle knowledge the user needed to complete this task a list of the prototypes and their information was provided to the user beforehand. A three object fuel cycle was constructed to represent the front end, reactor and back end of

a nuclear fuel cycle. The prototypes used and the required values can be found in appendix A.

Prototypes were constructed using the main view through either a drag and drop method or a text based click creation method. Then the required information was entered into their facility forms.

After the prototypes were created the user was tasked with creating two additional required objects for Cyclus to operate, a region and an institution. In both cases these Cyclus components were null or empty versions. This means that they did not require a form to be filled out (they have no inputs). To interact with these components users were exposed to the two views required to do this; a visual based region corral that allows user to create regions and move them around, and the institution view which is primarily a text based form.

Once these tasks were completed the user was informed that they had completed the hands on portion of the testing but were still capable of playing with the software if they'd like.

3.2.1c Questions and Comments

The final portion of the user experience testing consisted of four questions for the user followed by a period that allowed the user to make any comments about the software that did not come out during the use of the software.

One drawback of the user self-narration method is that when this questioning occurs the user is no longer in the testing phase and may have forgotten how they felt / what they

thought of a specific feature being asked about. To help reduce this possibility, the user was still free to use the software during the question and comments section.

The four questions were designed to answer specific questions the developers had about the software. Each was designed to give objective feedback to the developers by giving the testers a binary choice in answers. Questions which could result in subjective answers were rejected, however if a subjective answer was given after the binary answer was given it was still recorded.

The four questions asked of the users were as follows.

1. Of the two methods that allow for the creation of a new facility prototype which did you prefer?
    a. The two methods are text based versus drag and drop.
    b. The purpose of this question was to determine if users preferred one method over the other. This could be used to determine if one method would be implemented more often, or altogether replace the other.
2. Which view did you prefer, the region corral view, or the institution form view.
    a. This compares a more visual dragging field to a straight text based form.
    b. These two views provide very similar functionality both should have a very similar format. The purpose of this question was to determine which of format they should take.
3. Did the presence of user levels make filling out the forms easier for you?
    a. The purpose of this question was to determine if the presence of user levels provided a benefit to the user or if it was deemed detrimental.

4. Was it clear that when user level was increased, the values in each new form field were defaults?
    a. This question was asked to determine if the method of how default values are handled by CycIC was clear to the user or not.

3.2.1d Results

The results from the first round of user experience section are as follows.

Table 1 Time spent, and clicks used, on specific views during the first round of user experience testing.

View	Avg. Time (min)	Min	Max	Avg / Dev time	Avg. Clicks	Min	Max	Avg / Dev Clicks
Main View	5	3	9	2.39	80	60	120	2.46
Recipe View	2	1	3	3.88	25	12	38	4.51
Institution View	3	1	6	4.23	35	12	61	3.87
Region Corral	1	<1	2	3.83	10	5	14	3.36
First Form View	2	1	4	2.31	15	8	21	3.61
All other Forms	< 1	<1	1	1.82	6	5	9	2.0

It is important to note that a majority of the actions required to complete the task were done using the main view. The actions utilizing the form views are also included in the main view timing as well (This was done because the main view was open during this entire period in conjunction with the form views). This explains the abnormally high amount of time and number of clicks that are spent on this view compared to the other views.

The timing for each form was treated separately and a noticeable decrease in the time and clicks required to fill out a form was noted for all subsequent forms after the first.

This seems to indicate that as users became more familiar with the forms the difficulty of filling them out dropped with experience.

The time and click data here indicates that the region corral was an easier view to work with for most users. The average time required to work their way through the institution view was three times longer than that of the region corral. The average number of clicks was found to be 3.5 times greater in the institution view than in the region corral.

In addition to the time and click data, comments recorded from individuals during use of the software were tallied and Table II represents the comments that showed up most frequently (note, comments were grouped as long as their meaning was similar).

Table 2 Frequency of comments made during the user experience testing.

Comment	Frequency
What is the difference between a region and an institution?	9 (70%)
Institution View is confusing.	7 (53%)
Discovering Cyclus modules is easy.	6 (46%)
Drag and drop (DnD) capability is nice.	6 (46%)
New DnD prototypes should have a generic name.	4 (30%)
Linking facilities is confusing.	3 (23%)

The results here point to two main flaws in the CycIC system. The main issue raised was confusion over the difference between institutions and regions. Users were given a definition of these two objects however it remained unclear to them their purpose. Even though these are objects that exist in Cyclus and not creations of CycIC, a goal of CycIC

is to help users understand Cyclus. These results indicate that there is a deficiency in the CycIC software and more work needs to be done to improve understanding of the Cyclus system for users using CycIC.

The second major flaw resides in the institution view. 53% of the users felt that the institution view was confusing. This supports the results found in the time and click data that the institution view's layout and design is more difficult to use than that of the region view.

Two minor flaws came up in testing. First, new prototypes added to the simulation start without a name. 30% of the users found this to be confusing and preferred that the facilities start with a generic name. Second, the method of linking facilities via commodities through their forms was thought to be confusing by 23% of users. Both of these were determined to be good improvements to the software and will be modified for the second round of testing.

3.2.1e Questions Results

The results to the questions asked at the end of the user testing were as follows.

1. Of the two methods that allow for the creation of a new facility prototype which did you prefer?

Table 3 Results from question one.

Response	Frequency
Drag and Drop	10 (77%)
Text and Click*	3 (23%)

One thing to note here was that of the 3 responses in favor of the Text and Click method, 2 stated that they felt the Drag and Drop was better to learn the software on, but that the Text and Click would be faster once the user had experience.

2. Which view did you prefer, the region corral view, or the institution form view.

Table 4 Results from question two.

Response	Frequency
Region Corral	12 (92%)
Institution View	1 (8%)

The overwhelming preference here likely has to do with the previous discussed confusion and with documented evidence that general users prefer more visual interactions.

3. Did the presence of user levels make filling out the forms easier for you?

Table 5 Results of question three.

Response	Frequency
Yes	13 (100%)
No	0 (0%)

All users felt that the presence of user levels made filling out the forms easier. Additionally, users noted user levels had the added benefit of reducing clutter on the screen and preventing users from being overwhelmed with information.

4. Was it clear that when user level was increased, the values in each new form field were defaults?

Table 6 Results of question four.

Response	Frequency
Yes	13 (100%)
No	0 (0%)

There was no confused about the default values associated with different user levels.

3.2.1f Conclusions from Round 1

The results of the first found of user experience testing indicate some deficiencies within the CycIC code and provide feedback on what users would like to see in this graphical user interface in the future.

The first major issue is that users had difficulty determining the purpose of institutions and regions. CycIC is supposed to serve as the first interaction with the Cyclus code that most non-technical users will receive. This means that is must be able to inform users on the purpose and functionalities of Cyclus. The results indicate that CycIC is not accomplishing this goal in regards to institutions and regions. Therefore effort will be made to improve this functionality within the CycIC software.

The first change that will be conducted is to add additional tooltips to the region and institution views that will help to describe exactly how these two entities function. Additionally a much larger change will be made to the software by incorporating an additional feature to CycIC. This feature will be a new system that will provide users with ‘help’ will be implemented to the software that will provide information on how to use CycIC. While providing help on how to maneuver through CycIC is will explain to the user why certain actions are required by linking these actions to concepts in Cyclus.

The second major result from this testing is that the institution view as it stands is not the ideal way that institutions should be shown within CycIC. The amount of time and interaction that users spent on this view compared to the region corral, the comments made by users during the operation of the software, and the result of the second question all indicate that the region corral view is a superior method. Therefore the institution view will be reworked to make it more similar to the region corral.

The data indicates that the users tested in this round of user experience testing prefer more visual methods of interacting with the software than text based inputs. This supports other research that has been done in user experience testing [1] [2] [4]. This information will be used to provide users with more visual methods of interacting with the software in future version of CycIC.

Additionally the results from questions 3, and 4 showed that user levels were functioning the way that the developers had hoped when initially developing the software. The goal of this feature was to reduce the barrier of entry to users outside of the nuclear engineering community and to prevent clutter on the screen. The 100% affirmative response to question 3, and the additional comments made during question 3 indicate that this goal was achieved.

3.2.2 User Experience Testing Round 2

3.2.2a Testers

The first round of user experience testing was originally focused on graduate students with experience in nuclear fuel cycles. This round of testing will include persons outside the nuclear engineering field; potentially those with little to no engineering background.

The testers will be broken up into two groups; returning users and new users. The returning users group will be made up persons with experience using the software, and any user that participated in the previous user experience test. The new users group will be any tester that has no experience with the software. The two groups will be mostly identical in their tasks and questions, however the returning users will be asked to rank the changes made to the code since the previous round of testing.

3.2.2b Software Changes

The changes made to the code since the previous round of testing are focused on solving the problems that occurred most frequently in the first round of testing.

First, the institution view will be remade to make it more similar to the region corral view. This change is based on results from the first round of testing indicating that the region corral was preferred to the institution view. This will bring about several large changes to the software.

1. **Institution forms will open like region and facility forms**. This change is driven by the removal of form fields from the institution view. All institutions will now have forms separate from the institution view. These forms can be opened from the graphical objects representing institutions in the new institution view.

2. **Archetypes toolbar**. This change will add the ability to drag and drop institution archetypes to the simulation through an archetypes toolbar akin to that currently in the main view and region corral.

The region corral view will also be receiving an update. An unassociated institutions list will be added to the region view to insure that users are adding institutions to regions. This change is also aimed at helping to improve the user understanding of the Cyclus hierarchy.

Another change that will be made to the code will be the addition of a new method for linking facilities. The previous method felt unintuitive to some of the testers in the first round of UX testing. This new method will allow users to link prototypes through dragging a representative line from one prototype to another. Once the line connecting the two is completed a window will popup asking the user to choose which commodity will link the two facilities.

Finally new features are being added to the software to help users understand the functionality and structure of the Cyclus code.

1. **Tooltips.** New tooltips will be added to the region and institutions view to help users understand the function of these views and their underlying Cyclus archetypes.
2. **Help.** The presence of help windows was originally proposed for CycIC by was not implemented for the first round of testing. Like tooltips these help windows

will provide descriptions of how to interact with the object in CycIC, as well as how that object links back to the Cyclus core architecture.

3.2.2c Tutorial

As stated the second round of user experience testing on the CycIC software was conducted using persons with various different areas of expertise. As these users are brand new to both CycIC and Cyclus a user tutorial will be provided for the users before the testing began. The tutorial will be performed for the new testers/users immediately before the test is to take place. The tutorial will include both an overview of the Cyclus architecture and how to use the CycIC software.

The tutorial consists of a brief introduction to the concepts and ideas behind the Cyclus fuel cycle simulator. This included a description of the major components inside the Cyclus software; archetypes, prototypes, agents, regions, institutions, commodities, and recipes. Additionally the users are informed on how each of these pieces was connected. To aid in the understanding of how these components are featured in the CycIC software while these descriptions are occurring the person facilitating the user experience testing (the observer) will be demonstrating how each of the Cyclus components is represented inside CycIC.

The tutorial will also be available to any returning tester/user that asks for it.

3.2.2d User Experience Testing

The second round of testing will feature a more in-depth fuel cycle than the first round. The task will be to simulate a MOX fuel cycle using the architectures available in

Cyclus. The facilities and the inputs that the users will need to add to the simulation are found in Appendix A.

The reason that this test will be more in-depth is to provide the observer with more time to observe the user and additionally to open the user up to more interactions with the software. The previous round of testing was a limited test that was focused on specific views. This round of testing will be aimed at determining how well the software will perform in a typical use case.

This will present a challenge for the new testers, however the motions and actions that they must perform are fairly repetitive compared to the test performed previously. The exception to which is the addition of a reprocessing plant for separating plutonium, which will be new to returning testers as well. Because of this it is assumed that the new users will not need to be given a different task to complete.

During the test the observer will again be focusing on the amount of time that users are spending on each view. The amount of clicks made in each view will now be recorded by the software itself; this will provide a more accurate account of the number of clicks per view.

Additionally in this round of testing the observer will be recording the number of times that a user moves between two or more open views. This metric should help determine the amount of times users have to go back to previous view to recall a detail or to do something in one view that might be required in another. Reopening a view will be included in this count of switches.

Like in the previous round observers will be recording the comments that testers make and observations on how the user interacts with the software. The observer will also be on hand to answer any questions that the user might have, but as with the first round will try to limit the amount of interacting with the tester to reduce testing bias.

In this round of testing an observer will also conduct a 'perfect run'. The perfect run will represent the minimum amount of time and clicks required to complete the task given to the user. This will help be providing a baseline for time and clicks for the whole activity and for each action. This will provide information on where users are not interacting as efficiently as possible, which might indicate that there needs to be some work done on improving this section of the software.

3.2.2e Questions

The set of questions asked of new users and returning users will be mostly the same. However returning users will have an additional set of questions based on changes made to the software.

1. Repeat Testers
    a. Rate the following changes on a scale of 1 to 5 and provide reason if possible.
        i. Institution View
        ii. Help windows
        iii. Window Tooltips
2. New Testers
    a. Please Rate the following features on a scale of 1 to 5
        i. Form user levels

ii. Drag and drop to add new agents
iii. Test input to add new agents
iv. Institution View
v. Help windows
vi. Window Tooltips

3. All Testers
    a. Rate how the following views helped you understand Cyclus on a scale of 1 to 5
        i. Main View
        ii. Form Views
        iii. Institution View
        iv. Region View
        v. Help Windows
        vi. Tooltips
    b. Rate the functionality of the following form features on a scale of 1 to 5
        i. Drop down menus
        ii. Tooltips
        iii. Slider Bars
    c. What is one thing you'd like to see in the software that it is currently lacking
        i. This question is less for academic purposes and more for software development.

In this round of testing the questions asked of the users will be more focused on ranking the tools that exist in CycIC on a scale of 1 to 5. This will be done to improve the rigor of the test. It has been shown that ranking provides developers with a better method for determining the preference of features than open ended questions [2][3].

3.2.2f Results

A total of 15 persons were part of the second user experience test. The software developer also performed the same test. This was to give a best case scenario for how fast and efficiently a user could complete the task. The results of the test are seen in Table 7 through Table 12.

Table 7 Time spent on specific views during the second round of user experience testing.

View	Avg. Time (min)	Min	Max	Avg time / Developer time	Max Time / Min Time
Total Time	20.27	16.19	30.62	1.96	1.89
Recipe View	1.51	0.75	4.45	3.02	5.93
Institution View	3.48	1.92	5.37	3.70	2.80
Region Corral	0.96	0.45	1.20	3.80	2.67
Facility Forms	12.34	9.64	16.58	1.78	1.72

The timing results in Table 7 show that users averaged 20 minutes to complete the trial. This is roughly twice (196%) as long as it took for the code developer to complete the same task. A majority of the time spent was in the form views (Facility Forms in Table 7). Here again, users took roughly twice (178%) as long as the developer.

The most time spent on a class of child view was on the form views. This was expected as the scenario tested was much longer than the first test and involved filling out forms for nine facilities.

Based on the results from the timing test the facility forms performed the best out of all of the child views. The time ratio between the user average and the developer was smallest for the forms and the relative variation between users was also smallest. This

indicates that the facility forms are more consistently understood by users and the easiest to pick up for new users.

The difference between the developer times and those of the average user was much greater in the institution (370%), region (380%), and recipe (302%) views. This indicates that the learning curve for these views is likely higher than for the main view and the facility forms.

While the region view had 380% longer average user-to-developer time of interaction, the absolute amount of time spent was the smallest among all views as there were relatively few tasks to complete.. This view had the shortest time for the developer and required the least number of clicks (seen in Table 8).

Prior to their interaction with the region view, users had seen features similar to those in the region view multiple times. The difference here thus likely comes from having to check back to the problem statement to identify what to add to the region view and less from any confusion over the view. Because the developer time was so short here (0.25 min) any delay associated with consulting the problem statement will drive up the disparity in the times more considerably than views that took longer on average.

The institution view suffered from a large user vs developer time ratio as well as a large absolute time taken. User survey results indicate that this stemmed from confusion over how the deploy institution worked, specifically how to fill out the form.

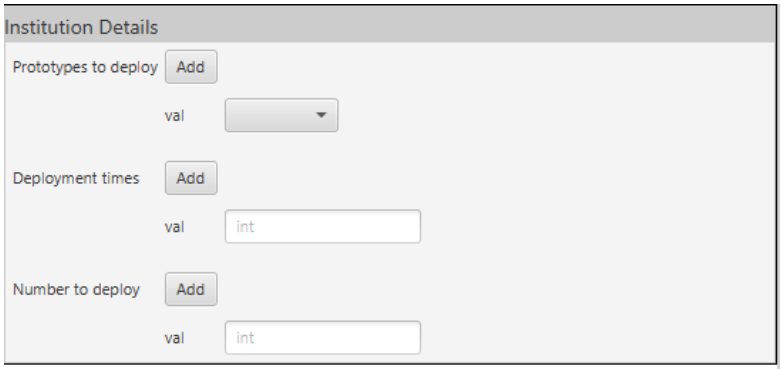

Figure 14 Form for institution view during the second round of user experience testing.

Figure 14 shows the form view that the users had to fill out for the deploy institution used in this test. In the view all three of these fields are shown as separate, however within Cyclus they are linked. In the Cyclus deploy institution the i^{th} member of each list is linked together, so that the i^{th} members of 'Prototypes to Deploy', 'Deployment times' and 'Number to deploy' all characterize different aspects of a single command issued to the deploy institution. Because CycIC was written to allow arbitrarily structured input fields, there is no way of linking input fields together. This could only be done by modifying the structure of the Deploy Institution's internal code. Something linking all of these fields together in the module like a map of pairs might provide a better visual link for example.

The recipe view's max time to min time ratio proved to be the worst of all of the views. This likely indicates that the view is intuitive to some and not to others. Given that the view took 302% longer on average for users to interact with relative to the developer, the recipe view should be subjected to additional scrutiny. Specifically the method for

creating and accessing recipes will be improved as it was a point of confusion for some users.

Table 8 Clicks used in a specific view during the second round of user experience testing.

View	Avg. Clicks	Min	Max	Avg/Dev Clicks
Main View	90	75	165	1.5
Recipe View	23	16	28	1.43
Institution View	26	21	29	1.44
Region Corral	6	5	9	1.2
Facility Forms	68	42	82	2.83

Table 8 shows that the behavior of the clicks mirrors that of the time spent on views. The reasons for this are the same as those listed in the timing section. The disparity between the user-to-developer completion time and number of click ratios comes from the fact that if a user is confused and needs to think about something they do not necessarily need to click the mouse. So while they may be spending time figuring out the next step they do not need to add more mouse interactions. In general users were much closer to the developer in terms of mouse clicks than time spent.

The one notable exception to this rule is the facility forms. Here users required 283% more clicks than the developer, but the time spent here was just 178% longer than that of the developer. This indicates that users may have felt like they had a good handle on the form views and did not need to think about how to use them, but still had to click their way through the views. In other words, while they understood the way to interact with the facility forms in general, the unique facility forms for each new facility they interacted with required a lot of clicking.

This is similar to the difficulties faced in the institution view. The forms may not perfectly reflect what the module developer was trying to accomplish computationally, because the module developer was not specifically thinking about how users might interact with the facility form while they were developing their code. This is one of the difficulties faced when developing graphical user interfaces for flexible software systems. An abstract module schema is similar to data when resolving semantic mismatches between how a developer things and how a user thinks. [13].

Table 9 Comparison between average time to developer time for UX tests 1 and 2

	Avg/Dev Time Test 1	Avg/Dev Time Test 2	Avg/Dev Clicks Test 1	Avg/Dev Clicks Test 2
Total Time	2.39	1.96	2.46	1.5
Recipe View	3.88	3.02	4.51	1.43
Institution View	4.23	3.70	4.12	1.44
Region Corral	3.83	3.80	3.36	1.2
Facility Forms	2.31	1.78	3.61	2.83

Table 9 compares the evolution the average time and number of clicks by testers relative to the developer from test 1 to test 2. There is a good improvement on the number of clicks required between the first and second tests. This suggests that there is an improvement in the code between the first and second tests with regard to the amount of physical interaction with the software the users need to accomplish a similar task. It also indicates that the learning curve for users with regards to clicks improves with repeated use of the software.

There is also a decrease in the timing statistics as well; however this decrease is much smaller. Specifically there is a slight decrease in the region corral and a small decrease in the institution view.

The lack of a decrease in the region corral may again be related to the limited amount of time that this view is actually interacted with. In this case the developer was recorded at 0.24 minutes. Additionally the reduced number of clicks on the region view seems to indicate that the time spent on the region view during the second round of tests is likely occupied by identification of the data to be entered into the view.

Table 10 Frequency of comments made during the user experience testing.

Comment	Frequency
How the deploy institution works is not clear.	9 (60%)
The tooltips take too long to load, the help popups seem more useful.	9 (60%)
The help menus answered a lot of questions I had.	8 (53.3%)
The institution view is much better now.	6 (40%)
The lack of units on the source facility is confusing.	6 (40%)
It would be nice if there was some sort of ‘okay’ or ‘save’ button on forms.	6 (40%)

Table 10 lists the top comments made by the testers. As with the last round of testing, comments that had similar meanings were coupled together, and only those comments that appeared more than 40% of the time were considered.

Confusion on how the deploy institution operated was the chief comment of users during the user experience testing. Most discussion of this can be found with the timing results for this test.

Next users felt tooltips took too long to load, and did not provide enough information for the amount of time they had to wait to see them for them to be worth it. Unfortunately this time is fixed by the programing language (Java) that CycIC is written in.

The addition of help menus seemed to help remove some of the confusion that occurred in the previous round of UX testing. The returning users approved of the changes made to the institution view between the first and second rounds of UX testing.

The lack of units on the source facility seemed to confuse some users. As with the deploy institution, the missing unit arises from structural features of the Cyclus code and schema which are aimed at allowing facilities and other modules to be as general as possible. In this specific case the current Source facility inside of Cyclus can be a source facility for anything. For instance, it could be a source facility for nuclear fuel, and the units of the source facility might be fuel assemblies rather than kilograms. The units of a facility are entirely up to the developer of the module. Again this sort of flexibility causes issues for GUI development [13].

Users would prefer to see a ‘save’ or ‘okay’ button to know that the actions that they have taken were reflected inside of the scenario they are making. The scenario automatically updates behind the scenes for users. This behavior seemed foreign to many users as it is counter intuitive to how they expected to interact with this type of software. The addition of ‘save’ and ‘okay’ buttons will be part of the next round of updates to CycIC.

3.2.2g Questions Results

Table 11 shows the response of users to specific features inside of CycIC. These tables asked users how they felt about some of the components of CycIC and also how these views helped them to understand Cyclus through CycIC. The ranking system was given to them as 1 – 5. 1 being the worst it could possibly be, 5 being the best it could possibly be.

Table 11 Test response to CycIC features.

Component	Average Response	Min	Max
Form User Level (New Users (NU))	4.03	3	5
Drag and Drop Agent Addition (NU)	4.25	3	5
Text Based Agent Addition (NU)	3.89	2	5
Institution View Changes (Repeat Testers (RT))	4.81	4	5
Help Windows (RT + NU)	4.23	3	5
Window Tooltips (RT + NU)	2.58	1	4

For repeat testers the main focus was the changes made to the software between the first and second user experience tests. The responses here were mostly favorable. The expectation to this was the tooltips. This was mostly due to the time it took for the tooltips to load and the limited information that they provided.

The institution view changes were approved by the repeat testers. This coincides with the comments made during the test and the improvement of the average time to developer time ratio between the two tests.

Of the new users the results were similar to those of the previous UX test. The drag and drop method was preferred over the text based input, and the form users levels

were viewed positively. However, using the rating system for this UX test highlights a deficiency of the previous testing round. In the previous round users were asked a yes or no question to indicate how they felt about the user level. While the user levels were viewed favorably again in this test, they only score a 4.1 on average. This result indicates that there is some room for improvement. Some users felt text based tier system made more sense; something more akin to 'beginner, intermediate, and advanced.

Table 12 Tester response to how helpful views were in understanding Cyclus

Component	Average Response	Min	Max
Main View	4.10	2	5
Form Views	4.67	4	5
Institution View	3.03	2	4
Region View	3.98	2	5
Help Windows	4.62	3	5
Tooltips	2.61	2	5

Table 12 depicts how well users felt that a view or component of CycIC helped them to understand the underlying purpose of that view within CycIC. In general users felt that the views did a good job with this. The two exceptions were the institution view and the tooltips.

While users felt that the region view was logical, the extra layer of the institution view did not immediately make sense to them. It was not clear from the view that the distinction between an institution and a region is which was responsible for 'building' facilities. It also confused users that an institution could not exist in multiple regions simultaneously. While both of these are limitations of the Cyclus code, it is important that

they be communicated to users. Most work will be done to improve this within the institution view.

The failure of tooltips came from what users felt was a lack of information given the amount of time it took for the tooltips to load. This is support by the results of the comments as well.

3.2.2h Conclusions from Round 2

The first major conclusion that can be drawn from the results from this test is that users that were taking the test for the second time felt that the improvements to the institution view were successful. Additionally they felt that help windows provided users with a good amount of additional information that was useful in working their way through the scenario.

However the addition of more tooltips did not fare well with new or repeated users. The major complaint here was that the tooltips took far too long to load and did not provide enough information to warrant this time. Instead they felt that the help popups offered more control and provided better information.

As with the previous round of testing the users felt that the drag and drop method of adding facilities to the scenario was the superior method for them, although in all cases they supported the existence of two options for adding new facilities.

Finally while effort was made to develop the capability of CycIC to inform users of how Cyclus works, there appears to still be a disconnect for many users. This indicates that more work needs to be done in this area such that new users are not left to struggle through building a scenario their first time. To improve this, the help view for the

institution view will be expanded to include further examples of the differences between regions and institutions and also the addition of an overarching Cyclus background will be added to the software.

BRIGHT-LITE

Bright-lite is software being developed as a Cyclus module to simulator a wide spectrum of reactors. It uses libraries that represent a specific reactor design to calculate the burnup, and the input and output isotopic vectors for a given input fuel. Bright-lite libraries are generated using Monte Carlo simulations for a reactor. While any Monte Carlo simulation is capable of generated the information required to make a Bright-lite library, OPENMC [14] and SERPENT [15] will be the supported software.

These libraries contain information about the fissile isotopes the reactor will burn, the parameters of the reactor used for the simulation, and libraries for each of the fissile isotopes used available to the reactor. For each isotope the following values are saved at predefined fluence steps; burnup, neutron production rate, neutron destruction rate, and a transmutation matrix.

A fluence based approach is used in conjunction with the neutron production and destruction rates of each fissile isotope inside the reactor core to determine criticality of the whole core [16]. A mass weighted sum of all of the fissile isotopes used in the calculation is created for the neutron production and destruction rates to represent the neutron production and destruction for a batch of fuel in the core. The neutron production and destruction of each batch in the core is then weighted by the flux profile of that batch. Finally all batches are summed to determine the total criticality of the core.

Additionally physics is available in the Bright-lite module as well. The ability to incorporate structural materials into the core allows for modeling of fuel cladding and

moderators. A disadvantage factor can be used to account for differences in flux between the moderate region and fuel regions [16].

Bright-lite operates in two modes. In the first mode, Bright-lite accepts a given fuel composition and then performs the necessary calculations to determine when the criticality of the reactor falls to one. The output of this first method is the burnup and output isotopic composition of the spent fuel. The second mode takes a desired burnup target and determines the composition of the fuel required to meet this burnup when the criticality of the reactor hits 1. The output of this method is the input and output isotopic compositions of the fuel.

4.1 Library Interpolation

4.1.1 Implementation

Libraries used in Bright-lite are specific to characteristics of a reactor, and to reactor type. This means there might exist a light water reactor library that was generated from 3% enriched fuel with a target burnup of 40MWd/kgIHM, and a library using the same reactor design with the fuel was enriched to 5% and a target burnup of 60MWd/kgIHM.

One method to increase the fidelity of Bright-lite would be to create libraries for all reactors with the values each of their parameters could take. This method would be computationally expensive because the list of parameters per reactor can be quite large; pitch size, thermal power, enrichment (or fuel composition if more than two fuel isotopes are present), burnup, pin size, reactor temperature, cladding type, moderator type, geometric dimensions of the reactor, etc.

Consider generating libraries for two such parameters with M values; for example burnup at values of 30, 40, 50, 60, 70, 80, 90, and 100 MWd/kgIHM (M = 8). The total number of combinations of parameters could be represented using the following formula.

$$N = \prod_{p=0}^{P} M_p \quad (1)$$

Where

p = a parameter

P = total number of parameters

M_p = number of values associated with the p^{th} parameter.

Table 13 Scaling of generating a library for each reactor parameter set.

Parameters	Values	Libraries Required
1	8	8
2	8	64
3	8	512
4	8	4096
2	4	16
2	8	64
2	16	256
2	32	1024

Table 1 shows how quickly the number of libraries required to describe a system of parameters becomes untenable to generate.

A much less computationally expensive technique for increasing the fidelity of Bright-lite is the use of a multivariate interpolation method. Interpolation allows for the creation of dynamic libraries to be used by Bright-lite to simulate a set of reactor parameters which does not have an existing library associated with it.

Interpolating on Bright-lite libraries poses several challenges. The first is that the interpolation must be multivariate. Each variable that is being interpolated upon might have its own scale and units associated with it. Therefore the interpolation scheme must be able to normalize the distance to the same scale for each variable.

The second problem is that the libraries in Bright-lite will not necessarily be at even intervals. Therefore the libraries will not form a multidimensional grid of points. This prevents Bright-lite from incorporating an interpolation scheme that requires a grid system to operate.

The two issues can be overcome using an inverse distance interpolation scheme. Bright-lite uses a modified version of Shepherd's Method [17] to perform the interpolations on the libraries that are required.

First all variables are normalized to a new scale from zero to one ([0, 1]). This is done by subtracting the minimum value from all values in a variable's range, and then dividing all of the resulting values by the total range the variable encompasses.

$$v_{i,new} = \frac{(v_i - v_{min})}{(v_{max} - v_{min})} \qquad 2$$

Where

i = the i^{th} isotope

v = value of the variable

This new scaling allows for distance comparisons between all of the variables being considered. The next step is to perform the inverse distance calculation to determine the new values for the newly created library. The function U(P) therefore represents the interpolated value given all points being used as reference for the interpolation.

$$U(P) = \frac{\sum_{i=1}^{N} F_i(\prod_{j\neq i} r_i^{\alpha})}{\sum_{i=1}^{N} (\prod_{j\neq i} r_i \alpha)} \text{ [18]} \quad 3$$

Where:

P = the series of coordinates for the interpolated point
N = total number of reference points
i = i^{th} reference point
r_i = the distance the i^{th} reference point is from the desired interpolated point
F = value for the i^{th} reference point
α = a scaling parameter

This formula is applied for all values of a library. For example for each value of burnup with regard to fluence, this interpolation is determined. Once this process is complete the library has the exact same structure as the largest reference library. This ensures it is capable of being used by the other functions of Bright-lite.

The α factor gives the interpolation some flexibility in how it behaves. Consider a 2-dimensional system with three points used to perform an interpolation. The three points chosen are (20, 3), (60, 5), (100, 1) and are marked in red in the following examples.

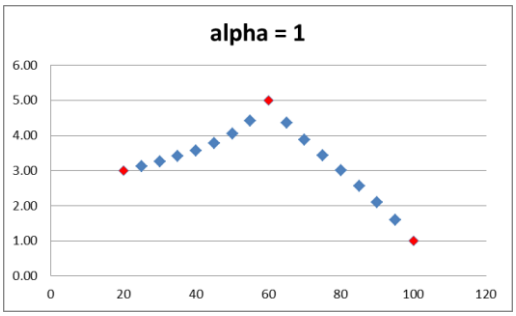

Figure 15 Two-dimensional interpolation, alpha = 1

The behavior does not go in a linear interpolation in this case because there are three points, and therefore each library still has some impact on interpolation. This is why there is slight bowing between the first and second points. Now consider higher values for α.

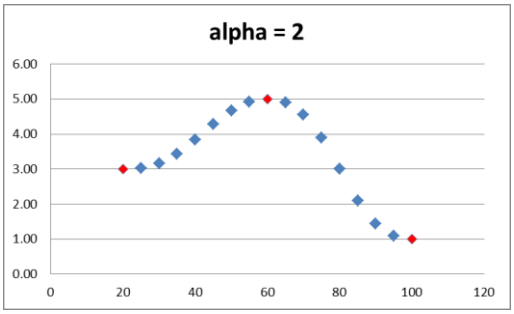

Figure 16 Two dimensional interpolation, alpha = 2

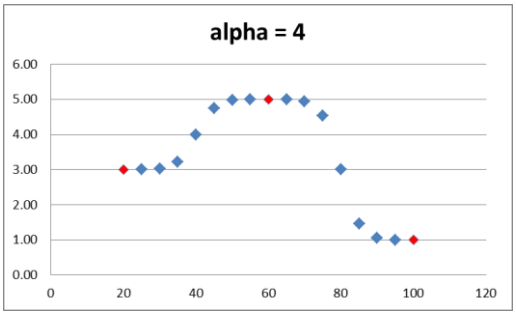

Figure 17 Two dimensional interpolation alpha = 4

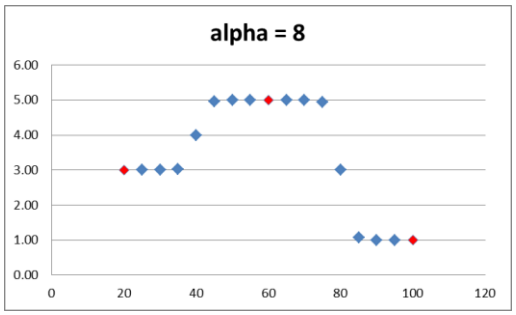

Figure 18 Two dimensional interpolation alpha = 8

As α increases the weighting toward each of the points used to perform the interpolation becomes stronger. As can be seen in Figure 18, once α becomes sufficiently large the method simply enforces the value of the closest known data point for the interpolation. In future work the sensitivity of the interpolation and Bright-lite results to alpha value will be more fully quantified.

Tests should be done to determine how well the alpha value performs over various distances. It is possible, that as the libraries used by the interpolator get closer together or farther apart from the interpolation point the correct alpha value to use will change. Determining the best alpha value for a given distance could help to improve the accuracy

of the method. To properly test this will require sweeping through the entire interpolation space between two libraries using a set alpha value. Then the alpha value will be changed and the same sweep will be performed. This will inform which alpha parameter behaves best over a large space.

4.1.2 TEST CASES

4.1.2a Blending Mode Reactor Cases

Testing the interpolation method using the blending mode of operation was conducted with three libraries. The libraries have the following characteristics.

1. Low LWR
    a. Enrichment: 2.2% U235
    b. Burnup: 20MWd/kgIHM
2. Standard LWR
    a. Enrichment: 3% U235
    b. Burnup: 33MWd/kgIHM
3. High LWR
    a. Enrichment: 5% U235
    b. Burnup: 50MWd/kgIHM

All schemes assume 3 batch fuel management. The first test uses the two bounding libraries (Low and High) to create a dynamic library. The objective of the test is to show that the dynamically generated library data calculates the correct fresh fuel enrichment to attain the targeted burnup value. The enrichment and discharge isotopics calculated by Bright-lite using the dynamic library will be compared against those obtained if the Standard library is used. The Standard library was not created by interpolation but

instead was determined from reactor physics calculations in the same way as the Low and High libraries.

Additionally the two other libraries (Low and High) will be run with the 33 MWd/kgIHM target burnup. These runs essentially assume that the one group cross sections relevant to the Low and High burnup/enrichment conditions, respectively, can be applied to the Standard case. The results demonstrate the accuracy of the dynamically generated library compared to the Standard library and give a comparison for the value added by the dynamic library.

For this case the objective will be to prove that the dynamic library is capable of matching the standard library on output composition of specific isotopes. The isotopes considered are; Pu238-242, Am241/243, and Cm242/244.

4.1.2b Forward Mode Reactor Cases

The forward mode operation test using the interpolator is conducted by inputting a fixed fuel composition into Bright-lite using the same three libraries used in the blending mode cases. In this mode of operation Bright-lite calculates the discharge burnup and output isotopics. The fuel is 2.5% enriched uranium and the burnup value and output compositions of each library will be recorded. Again a dynamic library is created using the Low and High libraries.

In this case the objective will again be for the dynamic library to match the standard library in the isotopes mentioned above. In addition to the isotope comparison the

discharge burnup of the fuel will also be used as a comparison measure. Again the goal will be for the burnup of the dynamic library to match that of the Standard library.

4.1.2c Multi-dimensional Case

This test verifies the accuracy of Shepard's method for use with interpolation of multiple libraries over multiple parameters.

To demonstrate that Shepard's method is capable of handling these cases when coupled with Bright-lite's reactor libraries several libraries were used to interpolate across their bounding parameter space.

Three libraries are interpolated upon and tested against a forth for testing, have the following characteristics.

1. Enrichment: 5%U235, Burnup 60 MWd/kgIHM
2. Enrichment: 7%U235, Burnup 100 MWd/kgIHM
3. Enrichment: 9%U235, Burnup 100 MWd/kgIHM
4. Test Value: Enrichment 6% U235, Burnup 60 MWd/kgIHM

Previous tests were integral evaluations in the sense that they compared fuel blending, burnup and criticality results, all of which depend on multiple data libraries acting together. For this test the purpose is to visualize the outcomes of the multidimensional interpolation on a single library parameter as well as to compare interpolation outcomes for that parameter alone. Hence only the neutron production rate (in n/s/unit flux) the start of reactor operation from each library is interpolated upon.

The objective for this test will be for the dynamic library to match the neutron production of the test library using the other three libraries as the libraries to interpolate upon. Additionally, this test will show how the interpolator constructs a interpolation space over the bounding limits of the parameters used by the interpolator.

4.1.3 RESULTS

The results of the blending case can be seen in Figure 19. In this and subsequent figures, isotopic masses at discharge are compared. Results are given as percent difference relative to the value obtained if the pre-calculated 'Standard LWR' cross section set is used.

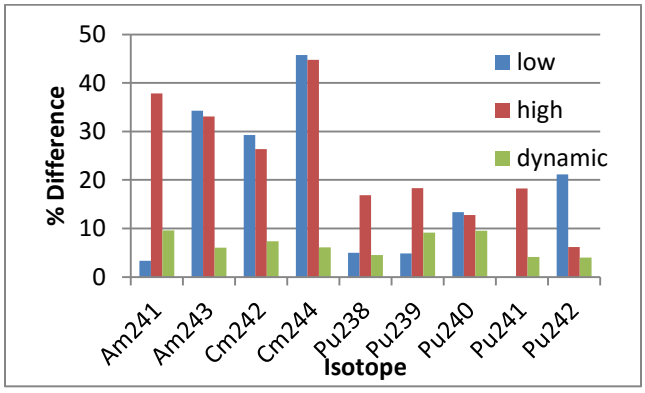

Figure 19 Results of Blending mode interpolation test

The generally large errors when the 'low LWR' and 'high LWR' cross section sets are used are unsurprising, since these sets are meant to apply to different initial enrichments and discharge burnups. However, the dynamically generated library obtained by using Shepherd's method interpolation of these two sets yields very good

results. Specifically, the interpolation method produced results that are within 10% of the target library's values. Hence the dynamic library also showed good improvement over using either the low or high libraries to model the targeted parameters (3% U235, 33 MWd/kgIHM).

In addition to tracking the isotopic compositions of the spent fuel, the calculated U235 enrichment of the fresh fuel was also recorded for each case. The results listed in Table II show that using the interpolated, dynamically generated library results in a calculated initial enrichment requirement that almost exactly matches that obtained if the 'Standard LWR' library is used.

Table 14. The U235 enrichment values for the reactors input fuel

Enrichment	Value	% Diff
Standard	3	0
Low	2.9	3.33
High	3.76	25.3
dynamic	2.98	0.67

4.1.3a Forward Mode Reactor Cases

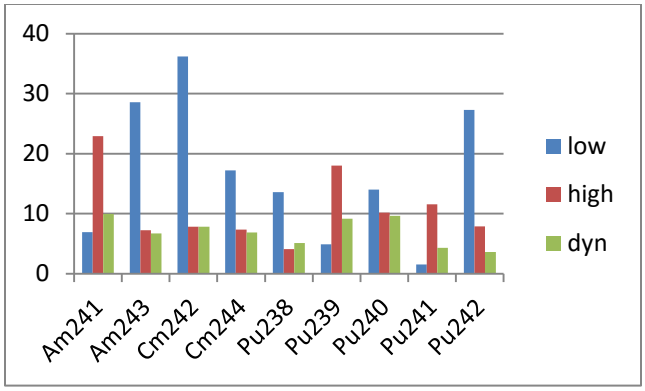

Figure 20 Results of the forward mode interpolation test

Figure 20 shows that again the library interpolation method matched the target library to within 10% for all isotopes tested. There is also good improvement over using just the low or high libraries to simulate the targeted reactor parameters.

For this test the burnup achieved by passing the 3.0% enriched fuel to the reactor is calculated. Table 15 shows that interpolation method generated a library that was capable of matching the burnup of the standard library.

Table 15 the reactor burnup values given 3.0% U235 fuel.

Burnup	Value	% Diff
Standard	33	0
Low	34.36	4.12
High	29.81	9.67
dynamic	33.11	0.33

In both tests the difference in the accuracy between the isotopic compositions and the enrichment/burnup values comes as a result of the unique one group cross sections in each of these libraries. Because the absorption of neutrons is the only way for these

higher order transuranics to leave the system (either through fission or transmutation) any variance in their one group absorption cross sections can have a big impact on their concentrations in spent fuel. The unique behavior of the cross sections of each transuranic means they do not change equivalently (meaning they do not all increase or all decrease) with a change in the reactor parameters. This can be seen in figures XXX and XXX by the large range of accuracies in the low and high libraries.

This result and the result of the blending test are reasonable even though the target library is relatively distant (in enrichment and burnup space) from either of the two libraries used in the interpolation.

4.1.3b Multi-dimensional Case Results

The results of this test can be seen in Figure 21.

Burn up (MWd/kgIHM) \ Enrichment (%)	5	5.5	6	6.5	7	7.5	8	8.5	9
60	14.799	14.771	14.683	14.552	14.405	14.269	14.158	14.074	14.014
62	14.795	14.764	14.669	14.529	14.376	14.237	14.127	14.046	13.989
64	14.780	14.745	14.644	14.496	14.338	14.199	14.092	14.015	13.963
66	14.754	14.714	14.606	14.452	14.292	14.156	14.053	13.983	13.936
68	14.714	14.669	14.554	14.397	14.238	14.107	14.012	13.948	13.908
70	14.661	14.608	14.489	14.330	14.176	14.054	13.968	13.914	13.880
72	14.596	14.534	14.410	14.254	14.108	13.997	13.924	13.879	13.854
74	14.519	14.447	14.320	14.169	14.035	13.939	13.879	13.845	13.828
76	14.433	14.352	14.222	14.079	13.960	13.880	13.835	13.813	13.804
78	14.341	14.250	14.121	13.987	13.884	13.822	13.793	13.783	13.783
80	14.247	14.148	14.019	13.896	13.810	13.766	13.754	13.757	13.764
82	14.154	14.047	13.920	13.808	13.738	13.713	13.719	13.735	13.750
84	14.065	13.953	13.828	13.725	13.670	13.663	13.687	13.717	13.739
86	13.983	13.866	13.744	13.649	13.606	13.616	13.660	13.705	13.733
88	13.909	13.789	13.669	13.580	13.546	13.572	13.637	13.698	13.732
90	13.843	13.723	13.605	13.519	13.490	13.530	13.619	13.698	13.735
92	13.788	13.668	13.553	13.468	13.440	13.491	13.604	13.703	13.742
94	13.741	13.624	13.511	13.426	13.397	13.454	13.593	13.712	13.752
96	13.703	13.590	13.480	13.395	13.363	13.422	13.585	13.724	13.762
98	13.673	13.565	13.459	13.376	13.341	13.400	13.580	13.734	13.770
100	13.651	13.548	13.449	13.369	13.333	13.391	13.577	13.737	13.773

Figure 21 Results of the interpolation method of on the neutron production of U235 at the first fluence step

The advantage of using an inverse distance system becomes apparent by looking at Figure 21 Results of the interpolation method of on the neutron production of U235 at the first fluence step. An interpolation surface is created that is represented by the heat map.

The interpolated value at the same parameters (6% U235 – 60 MWd/kgIHM) as the fourth test library is approximately 14.683 n/s/flux. A comparison between this and the accepted value of 14.650 n/s/flux shows a less than 0.5% difference. This good agreement supports the method currently being implemented.

It should be noted that to isolate the effects of the interpolation approach the neutron production rate is being checked before the reactor fuel is burned.. This means that the neutron production comes only from U235. Therefore it is only dependent on the U235 cross sections; other factors, like the buildup of other transuranics, will cause the neutron production rate (and other neutron balance parameters) to evolve over the cycle. This evolution is not explicitly benchmarked here but was indirectly addressed in the integral benchmarks presented earlier.

4.1.4 Library Generation

Bright-lite is limited in scope by the number of libraries that it has at its disposal. By adding additionally libraries Bright-lite can more accurately model current and future reactor types. Software (known as XSGen) was developed jointly at the University of Wisconsin Madison and the University of Chicago to automate the generation of a library. XSGen [19] couples a Monte Carlo code (OpenMC [14]) for generating one group cross sections and a burnup depletion code (Origen2.2 [20]) to derive the burnup, criticality, and transmutation curves.

When coupled with the library interpolation system already inside of Bright-lite XSGen will allow for the generation of suites of reactor libraries that will ensure that the library interpolation method always result in an accurate dynamic library. An accurate dynamic library (one generated by the interpolation method) is one that matches a library that was generated using the XSGen method to within a margin of error.

The system will work by first generating a set of libraries that bound a parameter space, for example burnups ranging from 30 to 60, and enrichments ranging from 3% to 5%. A new library that bisects the bounding libraries is then created using XSGen. The interpolation method will then develop a dynamic library at the same point using the bounding libraries

If the dynamic library is within tolerance of the known library's data, the method will be assumed to be accurate at that distance. A dynamic library with data outside of tolerance indicates that the method will not work at that distance. The distance will then be cut using a bisection method and the test will be run again. This process will be repeated until the dynamic library is acceptable.

The aspects of each library that will be used to determine if the libraries are within tolerance will be neutron production rate, neutron destruction rate, burnup and specific isotopes in the transmutation curves. These specific isotopes are; Uranium (235 / 238), Plutonium (238-240), Americium (241 / 243) and Curium (242 / 244).

The process of the bisection method can be seen in Figure 22. The grid represents a two dimensional (therefore two variable) system. The first image represents the initial

interpolation test. If that test fails, the distance between the two known libraries is cut in half and the test is attempted again.

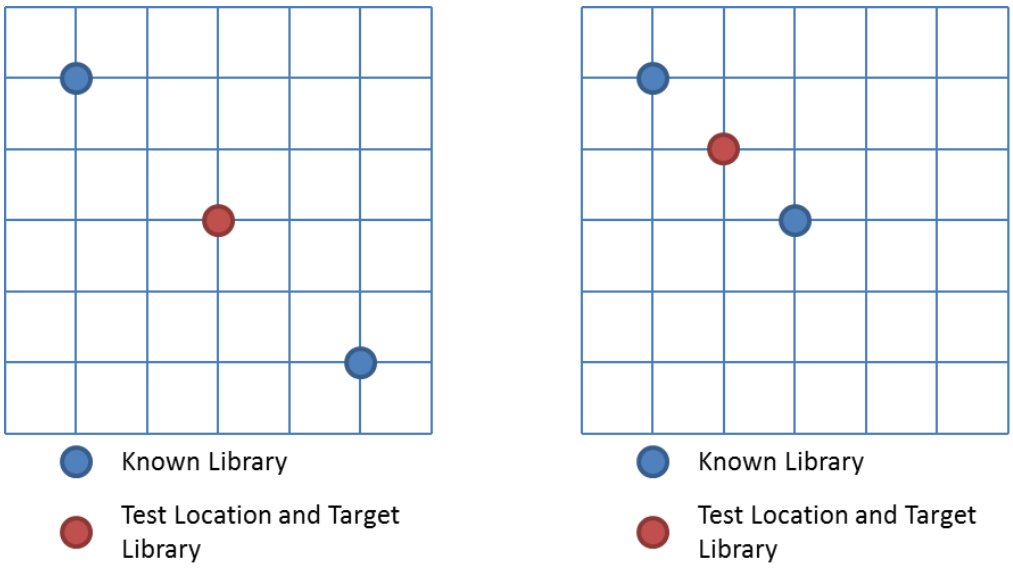

Figure 22 Depiction of the bisection method used in two dimensions. The left side represents the first test of the library interpolation tool. The second test depicts the adjustment made using the bisection method to repeat the test at a smaller distance.

Another interesting feature of this is that the alpha parameter can be used to help fine tune this library generation. During library generation alpha can be set such that the alpha is stored as a parameter of the library set and the user does not need to input it as a value.

4.2 Blending Methods

4.2.1 Two Stream Blending

The two mass stream blending problem aims to find the blending ratio of two streams of nuclear material whose compositions are fully specified at the isotopic level

which will attain some target or constraint associated with the burnup cycle. The cycle parameters subject to constraint and optimization within Bright-lite are criticality (multiplication factor must attain a target value), conversion ratio, fluence, and burnup. The constraints are generally specified as conditions on two of these factors: for example, the multiplication factor must be unity when a given burnup is attained, or the conversion ratio must achieve a target value at a specified fluence. Objective functions may also be defined. For instance, the blending problem may aim to maximize conversion ratio at a given burnup. However, if only two streams are available for blending, no more than one constraint or objective may be defined or else the system will be overdetermined.

In each method outlined below, Bright-lite first analyzes the two available streams to determine which of them is high-fissile and which is low-fissile. To determine which stream is the high-fissile stream the neutron production and destruction of the streams are used to determine the criticality of each stream at zero fluence. The classification facilitates the process of bounding the correct blending ratio.

4.2.1a Criticality Blending - Burnup Target

As mentioned, if only two streams are to be blended then a single constraint relating two of the cycle parameters must be used. For example given a target burnup the two fuel streams can be blended together to meet a criticality constraint of k=1 at fuel discharge. This is done using Bright-lite's forward burning method to burn two different compositions as starting points for interpolation on the correct blend. It is assumed that all of these functions behave in such a way that a root finding scheme can be used to determine the correct blend of the two streams. If the two initial composition guesses

indicate that the blending solution is going to exist outside of the physical limits on the composition ([0, 1]) then an error is thrown by Bright-lite to indicate that the fuels being used cannot reach be used to achieve the goals of the reactor.

A simple interpolation scheme is used to determine a blending fraction that will result in the burnup target. Since the constraints are assumed to be monotonic, a linear interpolation scheme is used. The following equation is used to perform the interpolation.

$$f_3 = f_f + \left(BU\left(stream_{nf}\right) - BU_{target}\right)\frac{\left(f_{nf} - f_f\right)}{\left(BU(stream_{nf}) - BU(stream_f)\right)}$$

Where

f_f is the fraction of high fissile material.
f_{nf} is the stream of low fissile material
BU(stream) is the result of the forward burnup calculation performed by Bright-lite
BU_target is the target burnup of the target fuel composition

The fraction determined from this method f_3 is then made into a new stream of material containing a weight fraction f_3 of the high fissile stream. This fuel stream is then again passed to the forward burnup calculation within Bright-lite, and the result is tested against the target burnup for the reactor. This convergence test takes the form of the following equation.

$$tolerance > \frac{abs\left(burnup_{target} - burnup_i\right)}{burnup_{target}}$$

If the target burnup is outside of the tolerance of the reactor the iteration scheme is performed again using the updated value. The system generalizes to the following equation after the first iteration.

$$f_{i+2} = f_i + \left(BU(stream_{i+1}) - BU_{target}\right)\frac{(f_{i+1} - f_i)}{(BU(stream_{i+1}) - BU(stream_i))}$$

This interpolation scheme is updated until the target burnup is converged upon. This behavior can be seen in Figure 23.

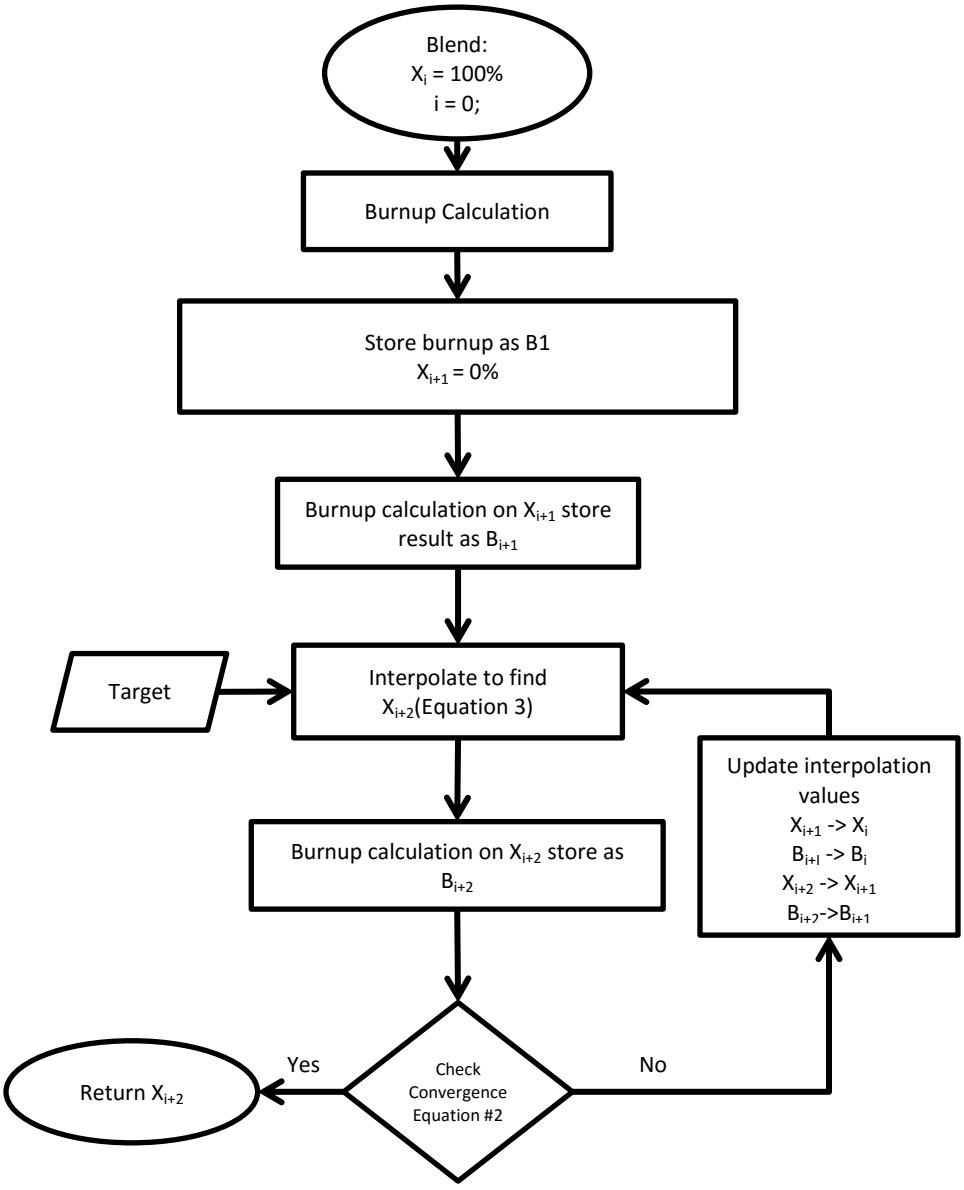

Figure 23 Two stream blending method flow chart.

This describes the method for imposing the criticality constraint at a target burnup. This method can also be used with the other constraints in most reactor systems. In some cases, though, the criticality/burnup constraint may not provide enough information to uniquely specify a two-stream blend. One such case is that of an accelerator driven system. Because an accelerator driven system keeps fuel subcritical, there is a constraint on the multiplication factor to never exceed 1, but many potential blends can satisfy this inequality. Therefore a criticality constraint on an accelerator driven system alone will not always ensure a solution.

4.2.1b CR Blending – Burnup Target

Similar to the method used to impose the criticality constraint at a target burnup, this system aims to meet a targeted burnup while matching a conversion ratio (CR) constraint. A CR may be defined for individual isotopes or elements, or groups of isotopes or elements. The conversion ratio is defined in terms of the number of atoms of these species created through nuclear reactions per atom consumed:

$$CR = \frac{atoms\ created}{atoms\ consumed}$$

Within Bright-lite, the list of isotopes which are used to calculate the CR may be customized by the user. However, Bright-lite's method of approximating the value of the CR works best when the group is large, e.g., all transuranic isotopes or all fissile isotopes. This is because of the manner in which Bright-lite's methodology for characterizing the transmutation of an isotope over time quantifies the buildup of fission products. Bright-lite tracks the fluence-dependent masses of fission products per mass of parent isotope,

but it does not differentiate between the parent isotope and its transmutation daughters when calculating fission product accumulations. Therefore, Bright-lite uses the following equation to approximate the fissile isotope CR:

$$CR = \frac{m_{FP,d} + m_{fissle,d} - m_{fissile,i}}{m_{FP,d}}$$

Where:

$m_{FP,d}$ = mass of fission products at discharge
$m_{fissile,d}$ = mass of fissile isotopes at discharge
$m_{fissile,i}$ = mass of fissile actinides at startup.

This makes the assumption that all fission product atoms are created via a fission event in one of the fissile isotopes.

A CR of greater than one implies that new fuel is being created faster than it is being burned, whereas a CR of less than one implies that fuel is being burned faster than it is created. These two situations cause the core to behave quite differently. For a CR of less than one it is possible to use criticality as a constraint for the reactor core because the core will eventually go subcritical. However, this may not hold true for a CR of greater than one. As fuel is being created faster than it is burned the multiplication factor of the reactor may increase over the reactor's lifetime (in practice, neutron poisons are used to keep the criticality of the core at 1 during operation). Therefore in this system the multiplication factor of the reactor may exceed one at fuel discharge. While the multiplication factor of such fuel would eventually drop below unity due to fission product buildup, generally this constraint does not determine cycle lengths.

It is still true, however, that a core must always be critical during its operational lifetime. Therefore an inequality constraint is imposed on the fuel that checks to make

sure the criticality of the core is always greater than 1. This is an example of an inequality constraint.

4.2.2 Multi-stream Blending Problems

Adding the possibility of a third possible fuel stream creates an issue for the method used in the two stream blending systems of Bright-lite. For each additional possible fuel stream being added another constraint must be imposed to ensure a unique solution to the blending problem.

In general this means that the number of constraints must be equal to at least the number of fuel streams minus one. These constraints must be equality constraints or objective functions which lead to an equality constraint. An equality constraint provides a target value for the parameter being constrained, while an objective function to be a minimized or maximized will generally give rise to an equality constraint. Inequality constraints, i.e. greater than (>) or less than (<) do not ensure unique solutions to the blending problem, but can ensure that a reactor will operate within a given set of parameters.

Two methods are being pursued to facilitate multi-stream blending within Bright-lite. The simpler method blends additional streams at a specified blending fraction into a new stream, reducing the blending problem to its original two-stream scope. The more general approach gives rise to a constraint based system that expands upon the method used in two stream blending.

For both methods, that sum of the blending fraction of each mass stream must equal one.

$$1 = \sum_{n}^{N} f_n$$

Where

N = the total number of mass streams available
n = the n^{th} mass stream
f_n = the blending fraction of the n^{th} mass stream

4.2.2a High Fissile Stream Blending

This method requires the user to input a targeted mass fraction for each additional possible fuel stream available to the reactor. These streams are then blended with the high fissile mass stream into a single fuel stream based on the targeted mass fractions.

$$S_{new} = \left\{\sum_{n}^{N} S_n * f_n\right\} + S_{fissile} * \left(1 - \sum_{n}^{N} f_n\right)$$

where

$S_{fissile}$ = high fissile mass stream.

S_n = n^{th} fuel stream.

f_n = mass fraction of the n^{th} fuel stream.

N = number of extra mass streams.

The fuel stream created using this method is then blended with the low fissile stream available to the reactor using the two stream blending methods discussed earlier.

This method works with all of the two stream blending functions defined above which increases the flexibility of each of these functions.

In practice the low fissile stream could also be used as the base stream that all of the material will be combined into, or additional mass streams with a criticality of greater than one can be combined with the high fissile stream and mass streams with a criticality of less than one can be combined into the low fissile stream.

A unique situation that arises due to this system is that it is possible that during the creation of the blended fuel stream, there will not be enough mass available to attain the targeted blend ratio. If this occurs, the user-specified blend ratio is adjusted to correspond to the amount of material which is in fact available. The check to ensure that enough mass is available is as follows.

$$Mass_n \geq Mass_{core} * f_n$$

An advantage of this blending method is that it generalizes to n streams quite easily. However it is limiting in the sense that it requires the user to input values for each of the fuel streams used beyond the standard high fissile and low fissile streams.

4.2.2b Multiple Constraint Blending

The ability to take more than one constraint (outside of mass constraints) allows Bright-lite a great deal of flexibility when creating reactor models. For example a user may wish to create a reactor fuel that reaches a desired burnup at a specific conversion ratio at a specific fluence. Again, here the number of equality constraints must be one less

than the number of mass streams (more equality constraints will over define the system, less may result in non-unique solutions).

This blending method leverages the two stream blending system that already exists within Bright-lite. Like the mass constraint system, the problem is framed as a system that contains multiple two stream systems that can be combined into a single stream. Under this approach, each stream n (out of N total streams) is represented by a normalized isotopic vector Sn and mass blending fraction fn as above. First consider the case where streams n=1 to n-1 have temporarily fixed mass blending fractions. Then stream n can be blended with streams n+1 through N to create a single new stream as follows:

$$S_{\{n,N\}} = \frac{1}{\sum_{j=n}^{N} f_j} * \left\{ S_n * f_n + S_{\{n+1,N\}} \sum_{j=n+1}^{N} f_j \right\}$$

where

S_n = isotopic vector of the n^{th} (out of N) fuel streams.

$S_{\{n+1,N\}}$ = isotopic vector of the stream created by combining streams n+1 through N

$S_{\{n,N\}}$ = isotopic vector of the stream created by combining streams n through N

f_j = mass fraction of the j^{th} fuel stream.

f_n = mass fraction of the n^{th} fuel stream.

All streams represent isotopic vectors which sum to unity. The mass stream S{n+1,N} is a combination of all mass streams from Sn+1 up to SN. S{n+1,N} can be represented with the following equation.

$$S_{\{n+1,N\}} = \frac{\sum_{j=n+1}^{N} S_j * f_j}{\sum_{j=n+1}^{N} f_j}$$

Where

n = the index of the n^{th} stream.

S_j = j^{th} fuel stream.

f_j = mass fraction of the j^{th} fuel stream.

N = number of mass streams.

Combining equations above, $S_{\{n,N\}}$ may instead be expressed as:.

$$S_{\{n,N\}} = \frac{1}{\sum_{j=n}^{N} f_j} * \left\{ S_n * f_n + \sum_{j=n+1}^{N} S_j * f_j \right\}$$

In the case that n = 1 this reduces to the expected result for the weighted sum of the isotopic vectors of every stream being blended.

$$S_{\{1,N\}} = \frac{1}{1} * \left\{ S_1 * f_1 + \sum_{j=2}^{N} S_j * f_j \right\}$$

$$S_{\{1,N\}} = \sum_{1}^{N} S_n * f_n$$

In the two stream problem, therefore, this resolves down to the following equation.

$$S_{new} = S_1 * f_1 + S_2 * f_2$$

Recall that in the two stream case there is only one unknown mass blending fraction. Because the blend fractions must all sum to one, f2 can be replaced with (1-f1). Therefore this becomes a problem with only one variable and therefore can be solved using the two stream blending method defined above.

$$S_{new} = S_1 * f_1 + S_2 * (1 - f_1)$$

By extension, for the n-stream blending problem, the general approach will be to fix the blending fractions of the first n-2 streams, let fn-1 be unknown, and note that

$$f_n = 1 - f_{n-1} - \sum_{i=1}^{n-2} f_i$$

The value of fn-1 which satisfies the first of the n-1 constraints is obtained. Then fn-2 is found by enforcing the second constraint and accounting for adjustments to fn-1 in order to keep the first constraint satisfied as well. While this algorithm generalizes to any number of streams, a sample implementation for the 3 and 4 stream cases is described next.

In the three stream case, then, f2 and f3 are first found using a guessed value of f1, and then f1 is subsequently determined by adjusting f2 and f3. The first step is carried out by performing two stream blending on streams 2 and 3 using one of the constraints provided to the system. Note that during this blending the streams S2 and S3 are only

allowed to blend over the bounds [0, (1-f1)]. An iterative process follows where the two stream blending on streams 2 and 3 is repeated each time a new blend fraction f1 for S1 is tested.

This type of system can be handled using a recursive algorithm. Therefore each time a stream is to be blended all of the streams that exist after it in the list of streams are also blended using other constraints. For example in a four stream system, S3 and S4 are blended using constraint 3(C3) to create S{3,4}. Then S2 is blended with S{3,4} using C2 to create stream S{2,3,4}. Note that each time S2 is blended with S{3,4} , S{3,4} must be recalculated. Finally S1 is blended with S{2,3,4} using C1. Again, each time S1 is blended with S{2,3,4} , S{2,3,4} must be recalculated.

One complication that arises because of this method is that the order that the constraints are implemented and the streams blended may matter. Therefore if a user supplies the constraints in one order it may cause Bright-lite to return an error that the fuel blending is not feasible, while if another order was chosen it may work fine. To account for this Bright-lite shuffles the constraints on a blending failure in an attempt to ensure blending is actually possible using the given constraints.

For example in a system of three mass streams and two constraints; k=1 (at discharge) and CR = 1.05. If two of the streams are non-fissile and only one is fissile, attempting to blend the two non-fissile materials to a criticality constraint of k=1 will fail as non-fissile streams are non-critical. It is, however, possible to blend them using a conversion ratio constraint. By shuffling the constraint order Bright-lite can avoid this problem.

The multi-constraint method assumes that values of the parameter being constrained will behave monotonically over the range of the constraint. Therefore, the methodology assumes that there is one unique value of the blending fractions that satisfies the constraints. For the constraints used in Bright-lite currently, this applies, however this method may fail when a function displays non-monotonic behavior.

Another limitation is the computational time it takes to solve this problem. This limitation is the major drawback of this method. While it is possible to extend the method to n fuel streams the process becomes very computationally expensive.

Each additional constraint adds another level to the blending process. The minimum number of blending/burnup calculations required to solve a one constraint problem is 2. For a 2 constraint system this expands to 4. Therefore the minimum number of steps required to solve a system of constraints goes as 2CN; where CN is the number of constraints. That's a representation of the minimum number of times the burnup calculation must be called. In reality each of the blending problems will likely call the burnup calculation at least 3 times. This means that number of burnup calculations that need to be performed would be 3CN, or XCN where X is the average number of blending attempts performed by the reactor during refueling (or startup) operations.

Hence, while the algorithm will be capable of handling n streams it will be limited by the maximum number of constraints being built into the Bright-lite system upon release (currently 3).

4.2.3 MINIMIZATION/MAXIMIZATION WITH EQUALITY AND INEQUALITY CONSTRAINTS

There are situations where reactors might be designed to minimize or maximize certain parameters. For example the goal of a reactor might be to minimize the CR while still hitting a target burnup (the desired situation for a reactor designed to burn transuranic fuel). To accomplish this Bright-lite utilizes objectives, which are defined as parameters that can be minimized or maximized by varying mass streams combinations. Objectives are implemented in a similar manner to the constraints are used in any of the blending methods currently in Bright-lite, with changes defined below.

Bright-lite is limited to one objective per blending. This is done eliminate ambiguity in how the objectives are being minimized or maximized. Using two objectives on the same problem does not ensure a true minimum or maximum for one of the objectives, as the values of the second objective solved will depend on the minimum of the first objective solved. Therefore when trying to solve two objectives the order in which the objectives are applied may give rise to unique and distinct solutions.

As before, there can be soft or slack inequality constraints. These constraints are to ensure that reactor systems are physical. One example of a slack inequality constraint is the criticality constraint referred to already, that is $k \geq 1$. Another example of a slack constraint would be to limit the fluence of the discharge batch to be < 2.0E25 n/cm2.

Because all of the constraints and objectives used in Bright-lite behave monotonically, for a system of two mass streams, maximization/minimization in the presence of inequality constraints becomes a matter of finding the blending fraction limits where the inequality constraints of the reactor are satisfied. For example, if the reactor is

given the inequality constraint that the discharge multiplication factor of the core must be greater than or equal to 1 (≥1), then the bounds on the blending fractions are those for which this condition is satisfied. These bounds are then evaluated to determine the value of the objective at each bound.

Continuing with the example of two-stream blending subject to the criticality inequality constraint, this is done by first setting fn equal to zero and then one (fn = 0 or fn = 1). If either of these fractions do not produce a reactor that will maintain criticality during the entire operational cycle then it means that there is some blending fraction for which k=1 is just satisfied, and some part of the domain for which the inequality constraint will not be satisfied. Thus, to identify this region, the end of cycle criticality is interpolated upon using the results from fn,0 and fn,1. This interpolation is done using the same scheme described in the two stream blending problem. If both fractions return failed criticality conditions Bright-lite returns an error to the user indicating the fuel cannot be used to operate the reactor. If both return k > 1, then any blend fraction will satisfy the inequality constraint and the optimal value will lie at one of the two blend fraction extrema.

Once the blend fraction domain which satisfies the inequality constraints is found, the value of the parameter being minimized or maximized is measured at the extrema of this domain. The maximizing or minimizing point becomes the streams blending fraction.

For a minimization:

$$f_n = \begin{cases} f_{upper}, & if\ O(f_{upper}) < O(f_{lower}) \\ f_{lower}, & if\ O(f_{upper}) > O(f_{lower}) \end{cases}$$

For maximization:

$$f_n = \begin{cases} f_{lower}, & if\ O(f_{upper}) < O(f_{lower}) \\ f_{upper}, & if\ O(f_{upper}) > O(f_{lower}) \end{cases}$$

Where

O(f) = the value of the objective at blending fraction f
n = the n^{th} mass stream
f_{lower} = the lower bounds of the blending fraction determined by the inequality constraints.
f_{upper} = the upper bounds of the blending fraction determined by the inequality constraints.
f_n = the blending fraction of the n^{th} mass stream

Non-monotonic functions are outside the scope of this approach; to handle them, the method would need to be generalized to finding all local minima or maxima that exist inside the bounds applied to the objective. As a rule, the inequality constraints available in Bright-lite will exhibit monotonic behavior with respect to the blend fractions. For example, one blending stream will generally offer a favorable neutron balance with an excess of neutrons produced per unit flux while the other will offer a deficit of neutrons produced per unit flux. Blending of these two streams will lead to monotonic behavior in the multiplication factor at the end of a burnup cycle.

Using objectives to blend only two streams in the absence of any equality constraints may not result in systems that are physically realistic or meaningful. An example of this would be attempting to reach a target burnup while minimizing fluence will result in a reactor that has the highest possible blend of fissile fuel. While Bright-lite will support the ability of users to do objectives with two stream blending, it is far more useful to use objectives with a larger number of mass streams and incorporate appropriate equality constraints.

Because Bright-lite's blending methods tackle only two stream problems at any given time (though the larger problem may be multiple streams) the objectives expand to n-stream systems just as the constraints would. For example a user may wish to create a reactor fuel that reaches a desired burnup at a specific conversion ratio while minimizing fluence as opposed to reaching having a specific fluence target.

For systems of more than two mass streams and therefore more than one constraint or objective, constraints are always applied to the blending problem first and the objective is applied last. This is done to ensure that all constraints are met when evaluating objective function values.

4.3 Results

4.3.1 Criticality Blending – Burnup Target

The blending technique was benchmarked against recipes from the VISION [3] software. The first case considered the VISION reference PWR; in VISION this is the 100_UOX_Once_Thru: UOX51 reactor. Bright-lite was given the inputs in Table 16 to match the appropriate VISION recipe.

Table 16 Input parameters for PWR case.

Parameter	Value
Burnup (MWd/kgIHM)	51
Cycle Length (months)	13
Batches	4

To benchmark the blending process the Bright-lite reactor was fed a stream of U235 and a stream of U238. It then determined the blend fraction of these two streams that would result in the targeted burnup.

The results of the blending calculation can be seen in Table 17. The values recorded in the table are the mass fractions of U235 and U238 in the fresh fuel supplied to the reactor.

Table 17 Input Composition for blending mode benchmark; Bright-lite vs VISION

Nuc ID	Bright-lite	VISION	% Difference
U235	4.28E-02	4.30E-02	0.47
U238	9.57E-01	9.57E-01	-0.02

The good agreement between Bright-lite and VISION here means that Bright-lite is capable of properly blending mass streams to meet a target burnup for this reactor system.

The output isotopic compositions are compared in Table 18. Included in this table are several of the major fission products and the plutonium isotopes of interest to fuel cycle simulation.

Table 18 Output isotopic compositions for the blending mode benchmark; Bright-lite vs VISION

Nuc ID	Bright-lite	VISION	% Difference
Cs135	7.18E-04	6.60E-04	-8.8
Cs137	1.80E-03	1.82E-03	1.0

Nuc ID	Bright-lite	VISION	% Difference
I129	2.86E-04	2.73E-04	-4.9
Sr90	7.64E-04	7.88E-04	3.0
Tc99	1.15E-03	1.14E-03	-0.6
U235	7.31E-03	7.65E-03	4.5
U238	9.22E-01	9.21E-01	-0.1
Pu238	3.34E-04	2.93E-04	-13.9
Pu239	5.96E-03	6.15E-03	3.0
Pu240	2.32E-03	2.91E-03	20.3
Pu241	2.06E-03	1.76E-03	-16.9
Pu242	8.85E-04	8.64E-04	-2.4

A majority of the isotopes showed a good agreement; the major exceptions being Pu238, Pu240, and Pu241. The discrepancies here are likely the result of Bright-lite's one group cross sections. Bright-lite does not utilize burnup dependent cross sections but rather a one group cross section that is representative of an entire cycle. This means that the effects of the changing concentration of important resonance absorbers such as Pu240 are not accounted for.

4.3.2 CR BLENDING – BURNUP TARGET

4.3.2a Benchmarking

The methodology for calculating conversion ratio in Bright-lite was benchmarked using the VISION [3] software. Two conversion ratios were tested to represent the types of reactors that Bright-lite is expected to model: a CR of 0.5 to represent a burner reactor, and a CR of 1.0 to represent a breeder reactor. The charge and discharge isotopic compositions were taken from the VISION recipes for equilibrium cycle of the fast reactor in a UOX to metal fuel fast reactor fuel cycle. (600 Metal Cooled FR at

equilibrium) The charge isotopic compositions were fed to Bright-lite and the discharge compositions were used as a tool for comparing the two models.

4.3.2b Burner Reactor Benchmark

The burner reactor was modeled in Bright-lite using an ORIGEN 2.2 library designed to simulate a fast reactor with a conversion ratio of 0.5 (FR50).

Table 19 shows the conversion ratio and burnup comparison between the two cases. Good agreement exists for both the burnup and the conversion ratio. The difference in the conversion ratio is affected by the shortcomings of the CR calculation method used in Bright-Lite (see Section 4.2.1b CR Blending – Burnup Target).

Table 19 Results of Bright-lite vs VISION burner benchmark: CR and Burnup

	Bright-lite	VISION	% Difference
Conversion Ratio	0.519	0.5	3.8
Burnup (MWd/kgIHM)	132.2	132.25	0.04

Table 20 shows the discharge isotopic comparison between the two methods. With the exception of two isotopes there is good agreement between the Bright-lite and VISION. The values reported are the mass fractions of the discharge fuel for several important transuranics.

Table 20 Results of Bright-lite vs Vision burner benchmark: discharge isotopic composition

Nuc ID	Bright-lite	Vision	% Diff
Am241	7.5E-03	7.6E-03	1.27
Am243	9.2E-03	9.2E-03	-0.46

Nuc ID	Bright-lite	Vision	% Diff
Cm242	6.7E-04	4.9E-04	-38.07
Cm244	6.9E-03	6.8E-03	-0.47
Pu238	9.6E-03	9.6E-03	-0.27
Pu239	9.7E-02	9.8E-02	0.57
Pu240	8.7E-02	8.9E-02	2.18
Pu241	1.6E-02	1.6E-02	-0.32
Pu242	2.7E-02	2.7E-02	0.35
U235	2.4E-04	3.6E-04	32.50
U238	6.0E-01	5.9E-01	-2.40

The difference in U235, while large, has only a minor impact on the overall results of the system because a DU blend stock is used and the total fraction of the fuel that U235 makes up is minor. Cm242 is in error because the half-life of that species is short (T1/2=162 days). Bright-lite's fluence-dependent libraries are constructed by irradiating unit masses of materials at a constant flux. This flux is chosen to be typical of the average flux in the reactor type to which the cross sections apply. But if the flux in the reactor being modeled is higher than that assumed to make the libraries, for example, the relative rates of radioactive decay and neutron interaction will be incorrect. Specifically, the ratio of interaction to decay rates will be higher. This is the case for the results shown in the table. The primary formation mechanism for Cm-242 is via capture in Am-241, which is evidently proceeding at a higher rate relative to Cm-242 decay than the fluence-based libraries assume.

4.3.2c Breeder Reactor Benchmark

The breeder reactor was modeled in Bright-lite using the same library as the burner reactor (FR50). While these two reactors represent different conversion ratios,

there was no available one group cross section library that modeled a fast reactor with a conversion ratio of greater than 0.5. But given that there are negligible neutron spectral differences between fast reactor systems having the same coolant, the effect of this was expected to be minor in regards to the overall reactor system. This expectation was borne out by the results.

Table 21 shows the conversion ratio and burnup of each method match with good agreement. Again the error in CR is affected by the shortcomings of the method used to calculate CR described in section XXX.

Table 21 Results of Bright-lite vs VISION breeder benchmark: CR and Burnup

	Bright-lite	VISION	% Difference
Conversion Ratio	0.9769	1.0	2.3
Burnup (MWd/kgIHM)	68.83	69	0.2

Table 22 shows the discharge isotopic comparison. The values recorded are mass fractions of the discharge fuel for some of the major transuranics.

Table 22 Results of Bright-lite vs Vision breeder benchmark: discharge isotopic composition

Nuc ID	Bright-lite	Vision	% Diff
Am241	7.5E-03	7.6E-03	1.27
Am243	9.2E-03	9.2E-03	-0.46
Cm242	6.7E-04	4.9E-04	-38.07
Cm244	6.9E-03	6.8E-03	-0.47
Pu238	9.6E-03	9.6E-03	-0.27
Pu239	9.7E-02	9.8E-02	0.57
Pu240	8.7E-02	8.9E-02	2.18

Nuc ID	Bright-lite	Vision	% Diff
Pu241	1.6E-02	1.6E-02	-0.32
Pu242	2.7E-02	2.7E-02	0.35
U235	2.4E-04	3.6E-04	32.50
U238	6.0E-01	5.9E-01	-2.40

There is good agreement in the values for U238 and the plutonium species; however the agreement for the other transuranics and U235 is poor. Similar to the burner case the concentration of U235 is lower in the Bright-lite case than the VISION case.

The Cm242 fraction is off due to the reasons previously discussed in the burner case.

4.3.2d Full Range Vision Benchmarking

A sweep of conversion ratios from 0.2 to 1.0 was conducted here to ensure that Bright-lite was capable of matching VISION through VISION's conversion ratio range. The same VISION and Bright-lite libraries used in the previous examples are used again here. For each conversion ratio tested the goal was for Bright-lite to hit the target conversion ratio, and burnup of the VISION recipe.

This test was conducted by inputting the recipes that VISION has for the conversion ratios tested into Bright-lite's forward mode. This means that Bright-lite burns the fuel until the criticality of the system is equal to 1. The criticality condition applies to all systems with a conversation ratio of 1 or less (while a CR of 1 should be breeding in as much fuel at the same rate it is burning fuel, the addition of neutron poison fission products will drive the criticality to one eventually).

Additionally to match the VISION run the non-leakage probability used in Bright-lite was adjusted until the burnups matched. This way the isotopics in each system (Bright-lite and VISION) could be compared on even ground.

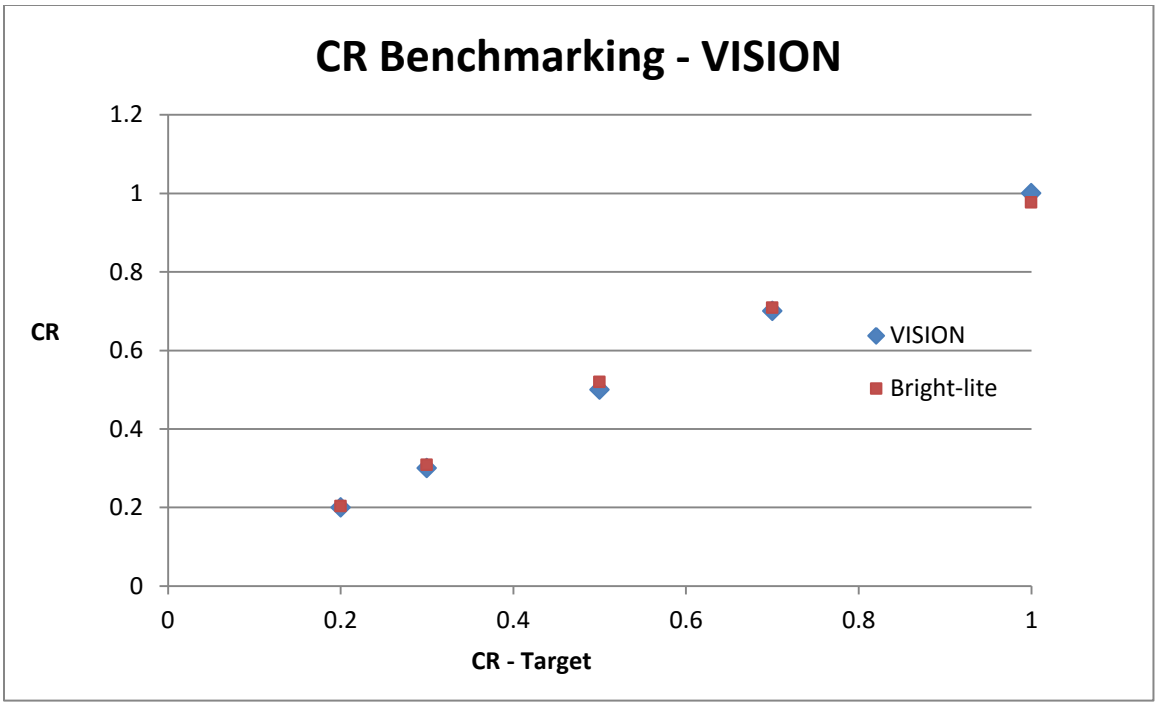

Figure 24 CR benchmarking through VISION's CR range.

Figure 24 shows a comparison between the CR for each VISION recipe and the CR returned by Bright-lite using the input isotopic compositions from that VISION recipe, while Figure 25 compares the discharge burnups.

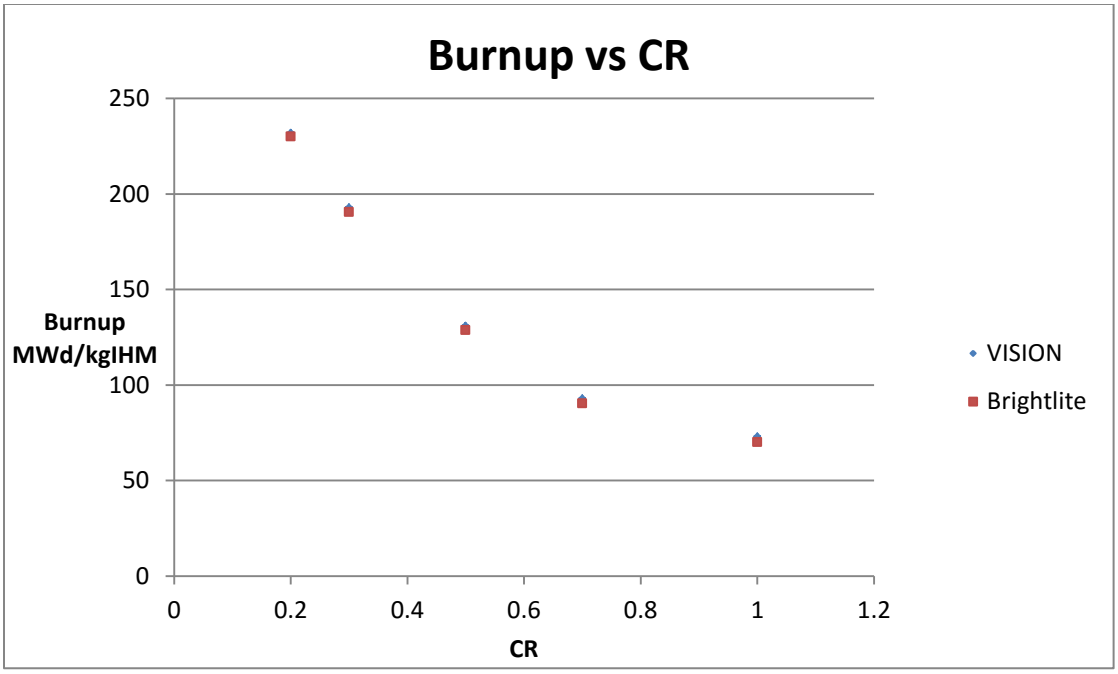

Figure 25 Burnup vs CR for Bright-lite vs VISION benchmarking test at equilibrium cycle.

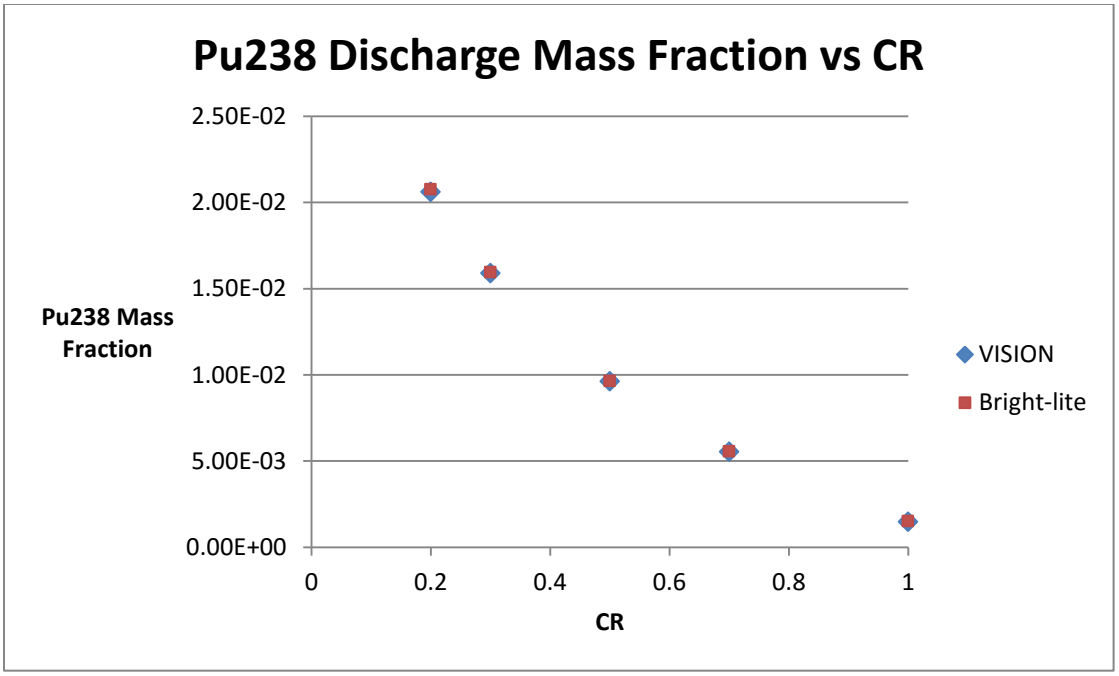

Figure 26 Discharge Mass Fraction of Pu238 during equilibrium cycle.

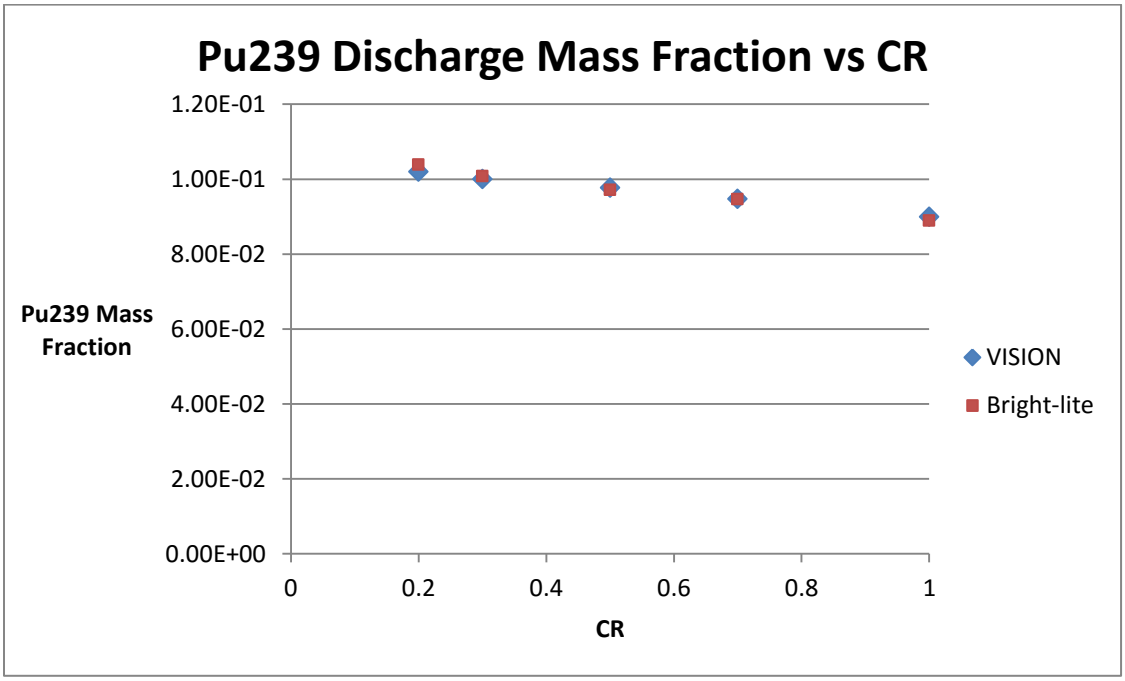

Figure 27 Discharge Mass Fraction of Pu239 during equilibrium cycle.

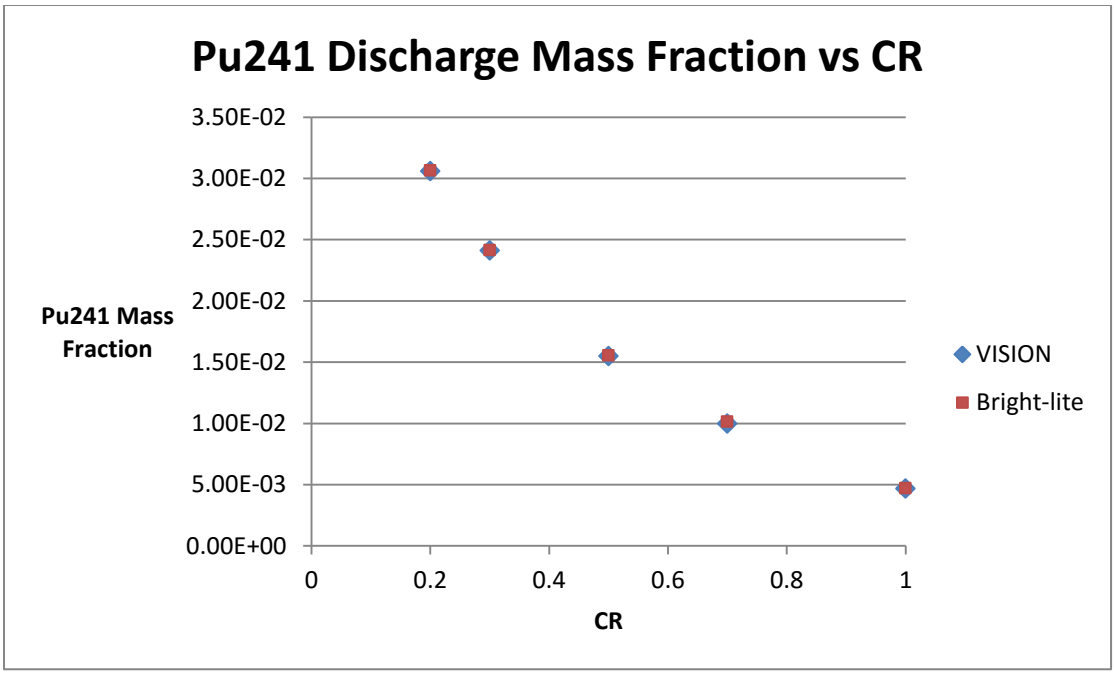

Figure 28 Discharge Mass Fraction of Pu241 during equilibrium cycle.

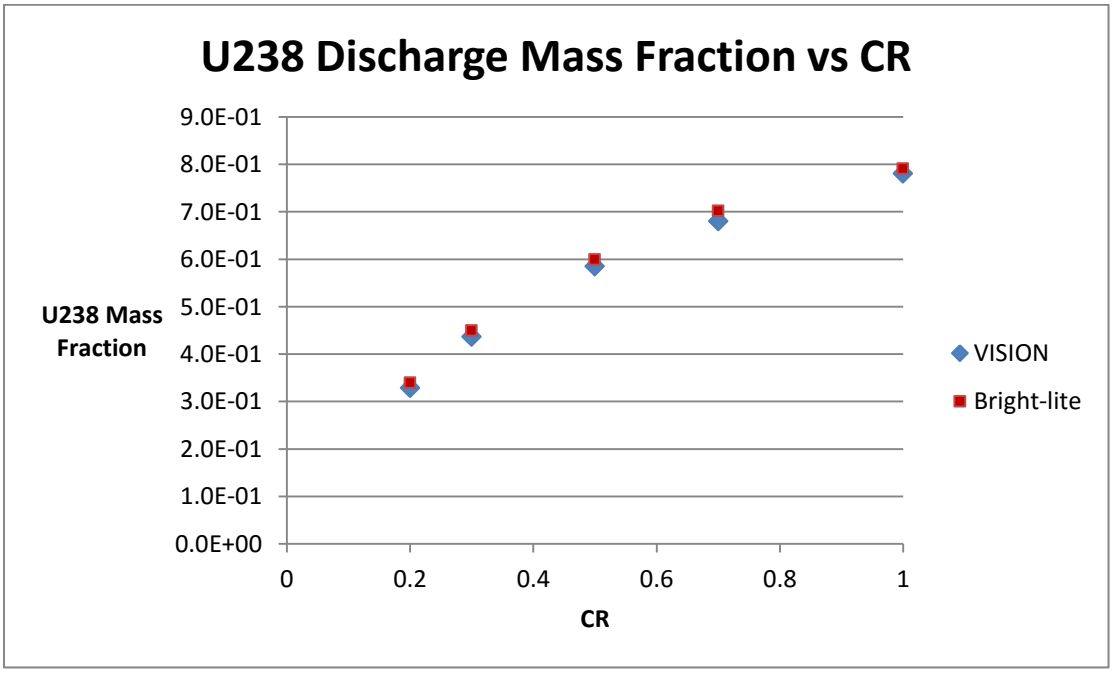

Figure 29 Discharge Mass Fraction of U238 during equilibrium cycle.

Figure 26 through Figure 29 depict the mass fractions of various isotopes at discharge. There is very good agreement through the whole span of conversion ratios tested affirming that Bright-lite is both capable of calculating the conversion ratio properly and matching the mass fraction of important isotopes in spent nuclear fuel.

4.3.3 HIGH FISSILE STREAM BLENDING

4.3.3a Burnup Blending with Depleted Uranium

To test the capabilities of the high fissile blending stream system for the three-stream blending problem Bright-lite was given three input streams: a U235 stream, a U238 stream, and a depleted uranium stream (0.25% U235, 99.75% U238). These streams were used to fabricate LWR fuel for a target burnup of 51 MWd/kg IHM. The addition of the DU stream serves to create a three-stream problem out of a two-stream system for which the correct U235/U238 blend is known. The mass fraction of depleted uranium is varied from 0 up to 80 percent of the high fissile stream (an additional run was conducted to determine the maximum DU stream fraction which was determined to be 96.1%). This caused the blending fraction for the high fissile stream to change but the actual amount of U235 in the system to remain constant to attain the burnup target of 51 MWd/kgIHM. The effects can be seen in Table 23.

Table 23 Tracks the behavior of the blending fraction of the fissile stream as more DU is forced into the stream using the high fissile stream blending method.

Forced DU Percent	Fissile Stream Blending Fraction	Mass Fraction U235
0	0.042	0.042
10	0.047	0.042
30	0.062	0.042

50	0.089	0.042
70	0.157	0.042
80	0.2553	0.042
96.1	1	0.042

The purpose of this test was to show that the high fissile stream is behaving as expected. Even though the input streams are changing the input fuel composition to the reactor is being held constant. This is the expected behavior from the system.

4.3.3b Transition Case: LWR to FR

A case that depicts the transition from a fleet of LWRs to a fleet of FRs is used to demonstrate that Bright-lite is capable of handling transition scenarios using the high fissile stream blending method.

In this case the fast reactors have a conversion ratio of 1.2. This allows for fast reactors to not only refuel themselves but also to build up an inventory of fast reactor spent fuel that can be used to start up new fast reactors.

Three light water reactors were started up at the beginning of the simulation. Then fast reactors began entering the system ~18 years into the simulation. The fast reactors were ramped up from there until the entire reactor fleet consisted of fast reactors. The reactor specifications are located in Table 24 and the deployment schedule for the reactors is in Table 25.

Table 24 Reactor specifications for this scenario.

	LWR	FR
Burnup (MWd/kgIHM)	42	55
Conversion Ratio	0.58	1.2
Core Mass (kg)	127,000	45,000
Batches	3	6

Core Thermal Power (MWt)	4,000	1500
Electrical Power (MWe)	1,320	645
Efficiency	0.33	0.43
Lifetime (years)	40	40

Table 25 Deployment Schedule for LWR to FR transition - High Fissile Stream Blending Method

Reactor Type	Number of Reactors	Month / Year
LWR	7	48 / 4
FR	1	200 / 16.6
FR	1	250 / 20.8
FR	1	300 / 25

The fast reactors employed the FR50 library. Bright-lite currently only works on homogenized geometries; therefore the breeding blanket and the driver fuel are blended to obtain a single input fuel composition. This makes the effective mass-averaged burnup lower than if the reactor were a fast burner reactor with only a driver region. The target burnup for these reactors was 55MWd/kgIHM [21], representing a mass-weighted average of the driver and blanket burnups.

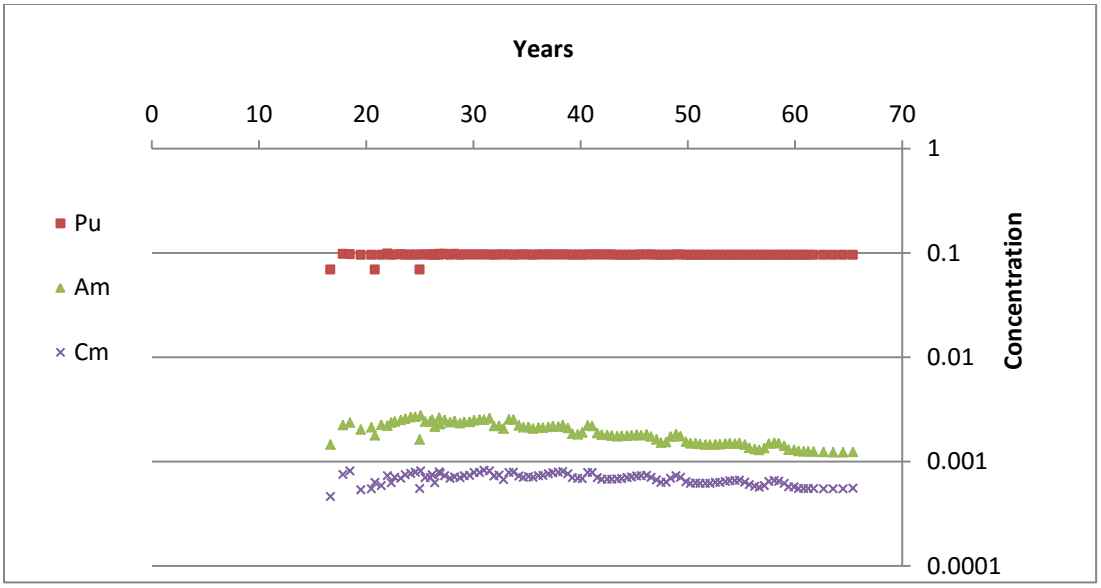

Figure 30 The fresh fuel composition of the fast breeder reactor.

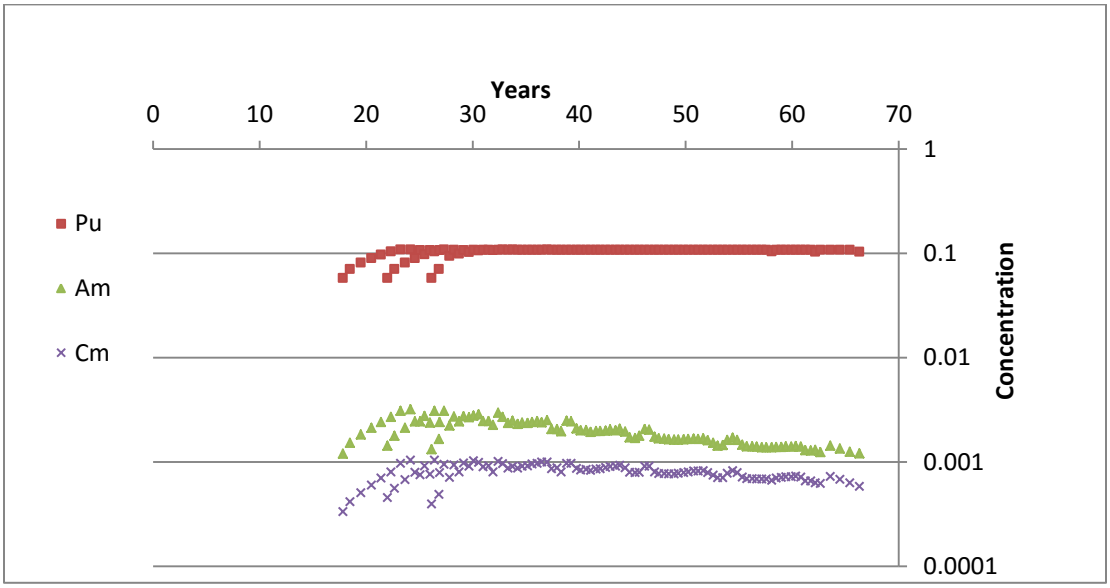

Figure 31 The spent fuel composition of the fast breeder reactor.

Figure 30 and Figure 31 show the actinide concentrations of the fresh and spent fuel for the breeder reactors. Each point on the figures represents one fuel batch. The batches associated with reactor startup and shutdown can be seen as outliers, with the startup batches having lower initial TRU composition than batches than subsequent reloads. The spent fuel shows a slight increase in the Pu concentration compared to the fresh fuel. This is to be expected for a system that is operating at a conversion ratio of 1.2.

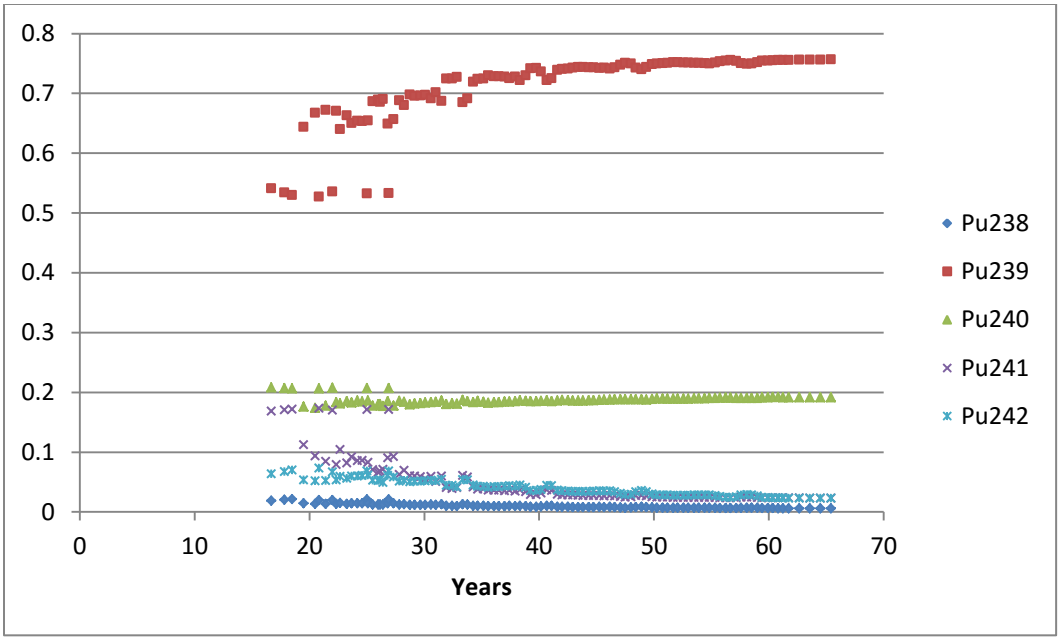

Figure 32 The concentration of Pu isotopes with respect to total Pu in the system in fresh fuel.

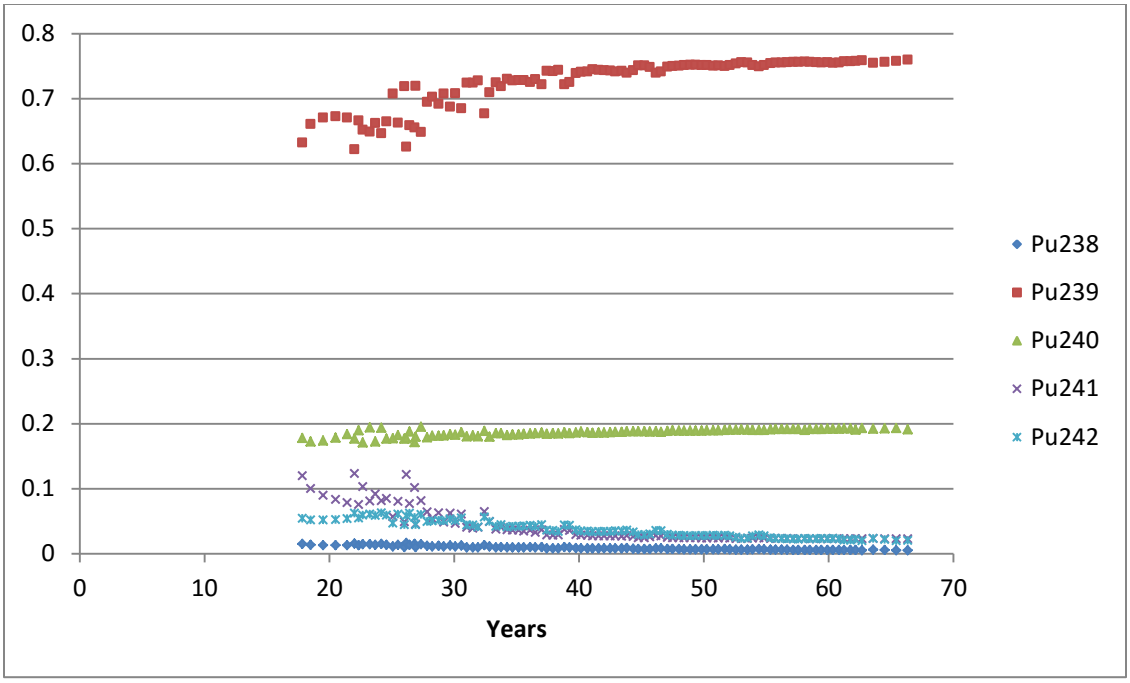

Figure 33 The concentration of Pu isotopes with respect to total Pu in the system in spent fuel

Figure 32 and Figure 33 depict the concentration Pu isotopes versus the total Pu concentration in the breeder reactor cores. These two graphs show that the isotopic vector of the Pu streams is staring fairly constant during a reactor cycle with the concentration of Pu239 and Pu240 increasing over time and the others decreasing over time. The plutonium being bred in each cycle in this fast-spectrum system has a significantly higher Pu239/Pu240 ratio than plutonium bred in the thermal spectrum LWRs. This is because the capture-to-fission ratio for Pu239 is significantly lower in a fast spectrum than a thermal spectrum.

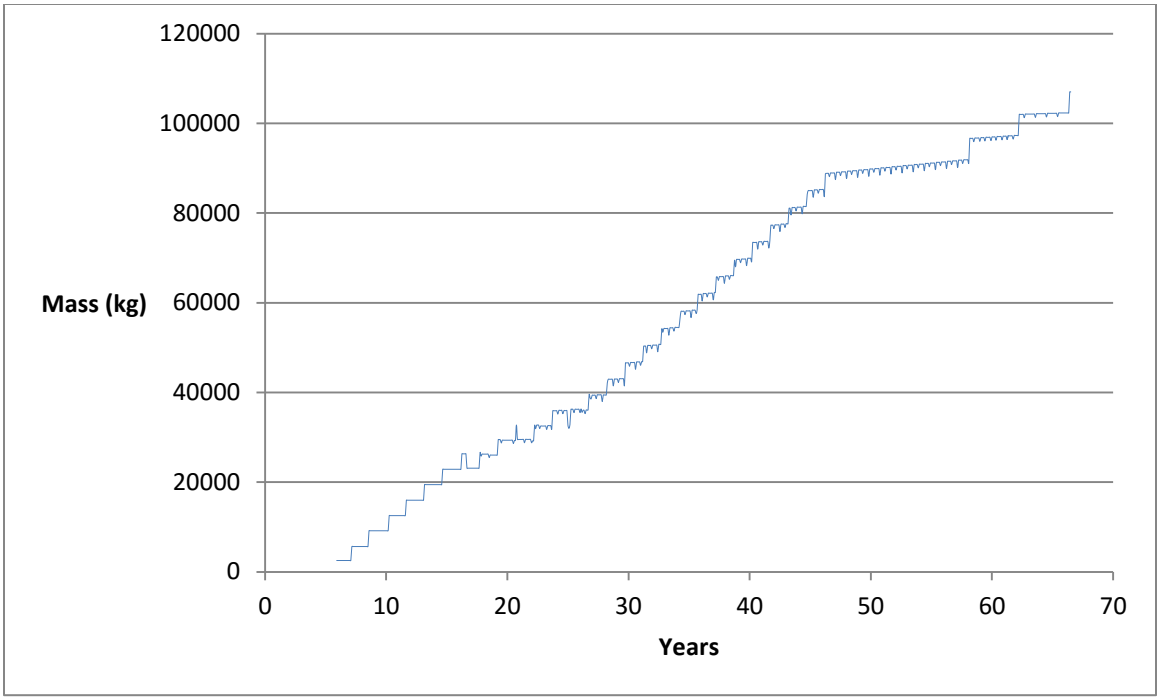

Figure 34 The amount of free transuranics in the system (that is the amount of TRU not currently in reactors).

Figure 34 shows the amount of free transuranics outside of reactors core over the entire simulation lifetime. Drops are caused by charge of TRU to reactors, while rises are caused by discharge of fuel batches from reactors. The total amount of transuranics is increasing because the LWRs continue to discharge TRU throughout their lifetime, only a limited amount of which is used to start the FRs, and the conversion ratio of the FRs is 1.2. The gain is steepest between year 30 and year 50. This is due to the presence of some fast reactors and the continued operation of the LWR fleet. Once the LWR fleet shuts down the steepness of the line reduces because the FRs are now the only facilities producing TRU.

4.3.4 MULTI-CONSTRAINT BLENDING

4.3.4a Benchmarking

The multiple constraints blending system was compared against the VISION recipes. To accomplish this, the VISION input recipe for a fast reactor at equilibrium was broken up into three components; DU, Pu, and Minor Actinides. The isotopic composition within each group is fixed to the values specified in the VISION recipe. The goal for this benchmarking is to prove that the blending system is capable of matching the VISION recipe using the constraints determined from the VISION reactor.

These three streams were blended to achieve a target burnup of 55 MWd/kgIHM, a conversion ratio of 1.0, with a discharge fluence target of 3.3E24 n/cm2 [22]. The fluence value here was chosen as because it is a representative value for fluence in fast reactors. While the burnup and CR correspond to an available VISION recipe, the actual value of the discharge fluence for in the VISION case is not available. The results presented here are somewhat sensitive to the discharge fluence target, but a third constraint to blend against is required by Bright-lite in this case. There must be two constraints for a three stream system, one of which is linked to the burnup target, and criticality may not be a reliable constraint for a fast reactor system with a conversion ratio of 1.0 or greater. Therefore fluence was chosen as the final constraint because typically material durability issues at high fluence limit the length of the fuel burnup cycle for fast reactors.

The resulting input isotopic vector for the fast reactor using this blending technique and the comparison to VISION can be seen in Table 26. The Bright-lite blending algorithm reproduces the VISION input composition with minimal deviation.

Table 26 A comparison between input isotopic vector generated by the multi-constraint blending method in Bright-lite and the VISION recipe.

NucID	Bright-lite	Vision	Percent Difference
TRU	3.18E-03	3.25E-03	**-2.32**
Pu	1.35E-01	1.34E-01	**0.22**
RU	8.62E-01	8.61E-01	**0.09**

4.3.4b Burner Case

This case features a fuel cycle aimed at reducing the transuranics produced by light water reactors using a fleet of burner fast reactors. The reactor specifications for the system are located in Table 27. All of the fields in are input into Bright-lite (with the exception of the CR for the LWR which is calculated by the software). The startup schedule for the reactors in this system is located in Table 28.

The objective of this test is to show that the blender proper blends the fuel to facilitate the removal of TRU from the system. This will be shown by the total amount of TRU available to the system decreasing as the fast reactors are brought online to burn the fuel.

Table 27 The reactor specifications for the burner scenario.

	LWR	FR
Burnup (MWd/kgIHM)	42	134
Conversion Ratio	0.58	0.8
Core Mass (kg)	127,000	45,000
Batches	3	6
Core Thermal Power (MWt)	4,000	4000
Electrical Power (MWe)	1,320	1720
Efficiency	0.33	0.43
Lifetime (years)	40	60

Table 28 The startup schedule for the reactors in the multi-constraint blending method burner transition scenario.

Time (months / years)	Reactor Type	Number of Reactors	Lifetime
50 / 4	LWR	6	40 years
400 / 33	FR	2	60 years
750 / 62	FR	2	60 years

The burner reactors in case were targeted to a burnup of 134 MWd/kgIHM and a conversion ratio of 0.8. Two constraints were imposed: the system multiplication factor must be at least 1.0 when the discharge burnup is reached, and the fluence at discharge cannot exceed 4E24 n/cm2. In practice, only the criticality constraint was active in this simulation.

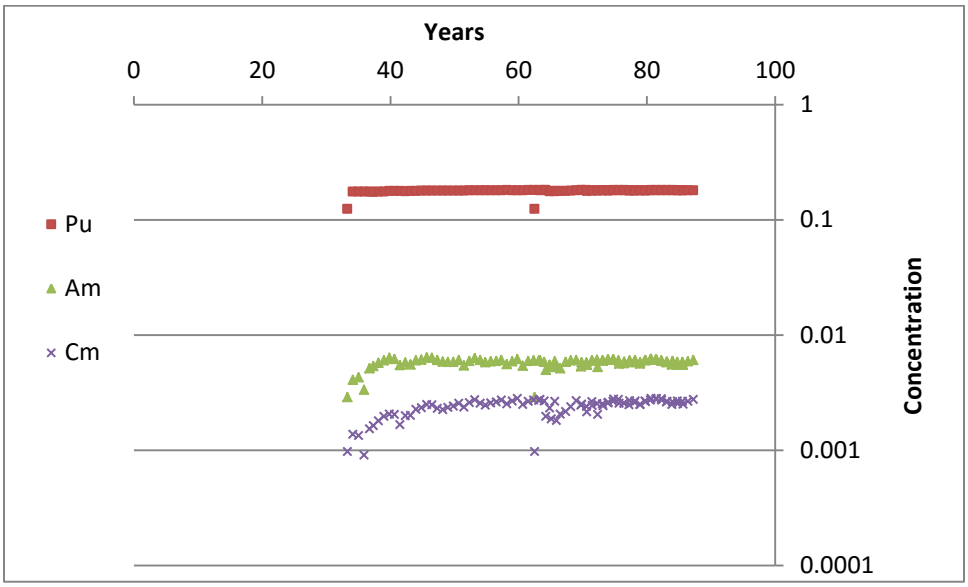

Figure 35 The fresh fuel composition of the fast burner reactor.

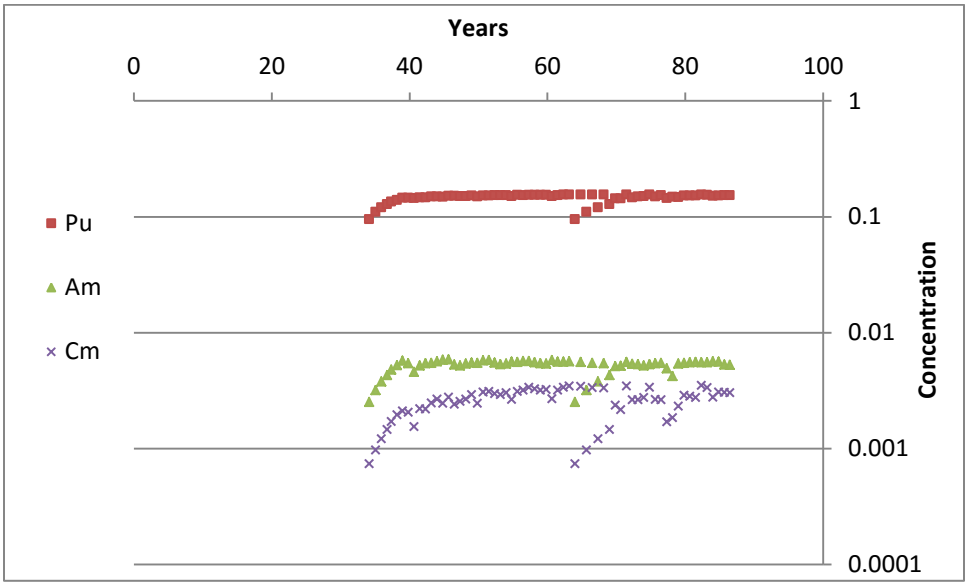

Figure 36 The spent fuel composition of the fast burner reactor

Figure 35 and Figure 36 depict the concentration of the actinides in the fresh and spent fuel for the fast burner reactors. As expected, discharge Pu, Am and Cm mass fractions are generally lower than charge mass fractions. Transients associated with the startup of the first and second waves of reactors can be seen. Specifically the two low points in all three actinide stream come from startup batches that contain lower concentrations of these compared to DU.

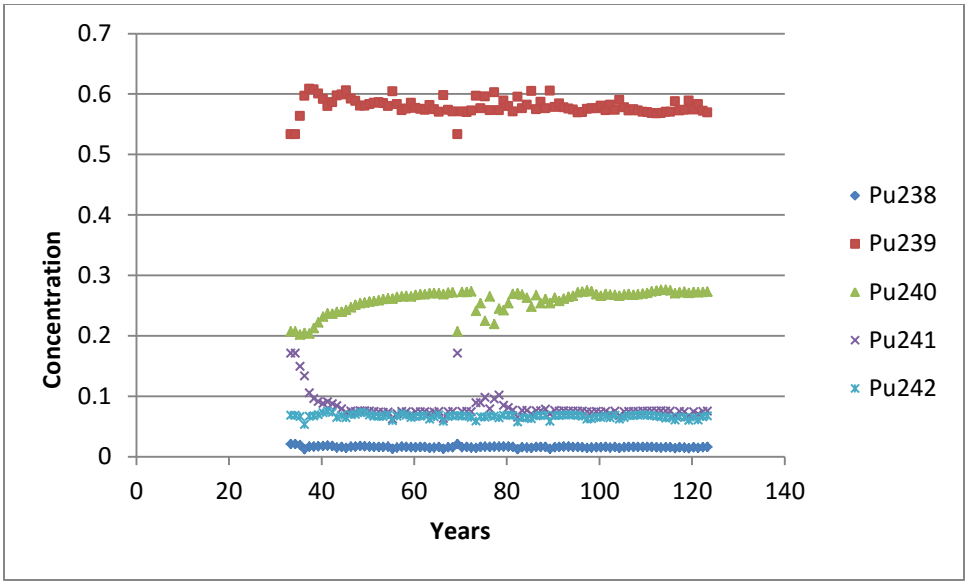

Figure 37 The concentration of Pu isotopes with respect to total Pu in the system in fresh fuel

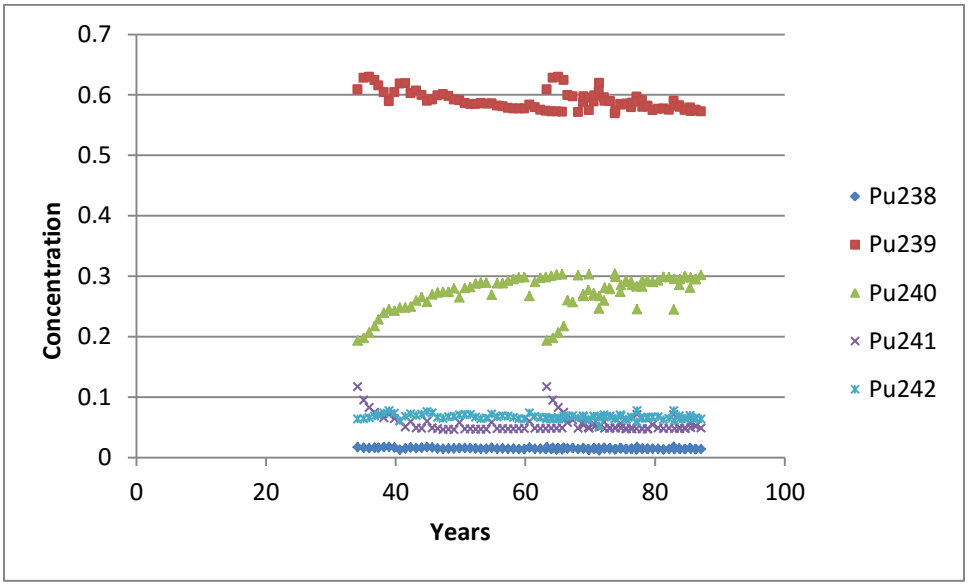

Figure 38 The concentration of Pu isotopes with respect to total Pu in the system in spent fuel

Figure 37 and Figure 38 show the plutonium isotopics at charge and discharge. Since the FR are gradually burning down their own Pu but also accepting Pu top-up from the LWRs, the Pu isotopics converge to a quasi-equilibrium where charge and discharge compositions are the same.

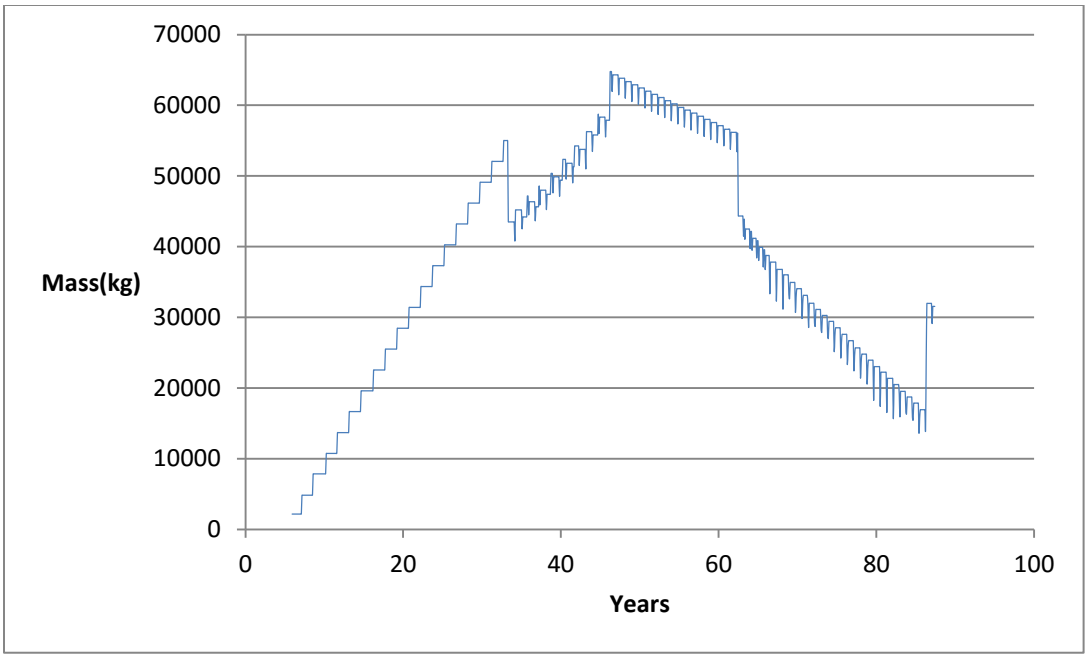

Figure 39 Total TRU in the system not locked up in reactors for the burner case

Figure 39 shows that the burner reactors are effectively reducing the amount of TRU being added to the system from year 30 to year 50 and reducing stored spent TRU after year 50. The drops in years 33 and 62 are caused by the startup of the FRs, while the jump in year 44 follows from the shutdown of the LWRs. The final spike in the graph comes from the shutdown of the first two fast reactors which had been commissioned in year 33. The other two FR would continue to run but the simulation ends in year 90.

4.3.4c Breeder Case

A breeder case was also run for the multi-constraint method. The LWRs in this case are the same as they were in the burner case in 4.3.4b Burner Case. The fast reactors used in this case maintain the same fluence target (3.3E24 n/cm2) used in the VISION benchmarking case mentioned in section, 4.3.4a Benchmarking, but have a burnup of 55 MWd/kgIHM [21], and a conversion ratio of 1.2. The lower burnup comes as a result of the blending of a breeder blanket and a burner region to make a homogenized core. Full reactor specifications can be found in Table 29.

The goal for this test case is to show that the blender is properly bleeding fuel with the CR of 1.2. The result of this will be a steady increase in the TRU inventory of the full fuel cycle. Additionally, a slight increase in the amount of plutonium between charge and discharge of the fuel should be seen.

Table 29 Reactor specifications for breeder case

	LWR	FR
Burnup (MWd/kgIHM)	42	55
Conversion Ratio	0.58	1.2
Core Mass (kg)	127,000	45,000
Batches	3	6
Core Thermal Power (MWt)	4,000	1500
Electrical Power (MWe)	1,320	645
Efficiency	0.33	0.43
Lifetime (years)	40	20

Unlike the burner case using criticality as a constraint on breeder reactor cycle lengths is not realistic. Since new fissile fuel is being bred in to offset the buildup of fission product poisons, a breeder whose cycle length were determined by criticality would exceed the realistic fluence constraints on the structural materials of a fast reactor core.

The deployment schedule for the reactors in the breeder case can be seen in Table 30.

Table 30 Deployment schedule for the reactors in the multi-constraint method breeder case.

Time (month/year)	Reactor Type	Amount of Reactors	Lifetime
50 / 4	LWR	4	40 years
200 / 16	FR	1	20 years
250/ 20	FR	1	20 years
300 / 25	FR	1	20 years

Figure 40 and

Figure 41 depict the concentration of the actinides in the fresh and spent fuel for the fast breeder reactor. These concentrations show that there is a clear increase in the concentration of Pu and other TRU between loading and discharge of the fuel.

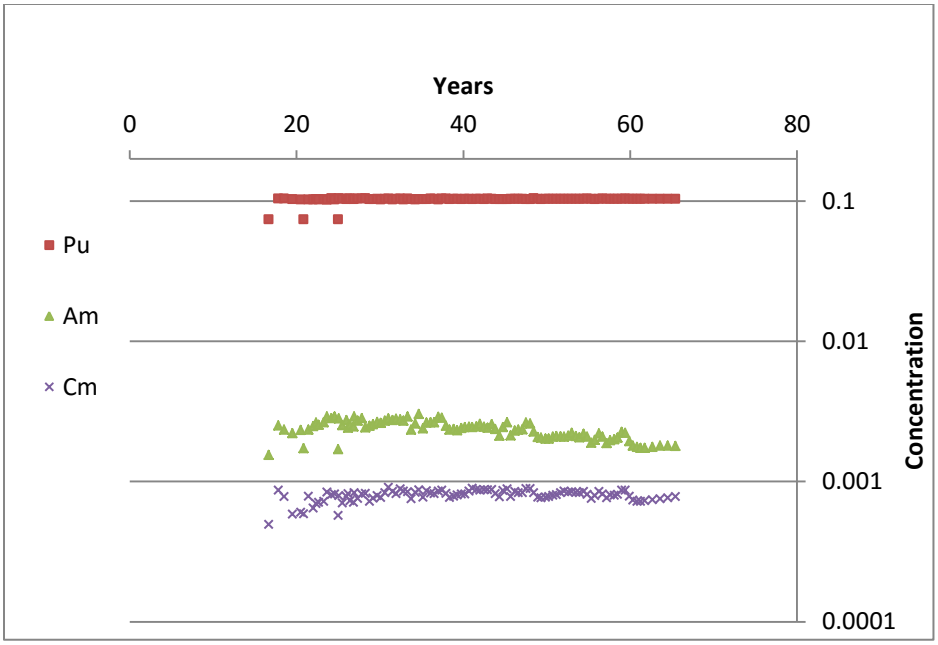

Figure 40 Fresh fuel composition for the multi-constraint blending method breeder case.

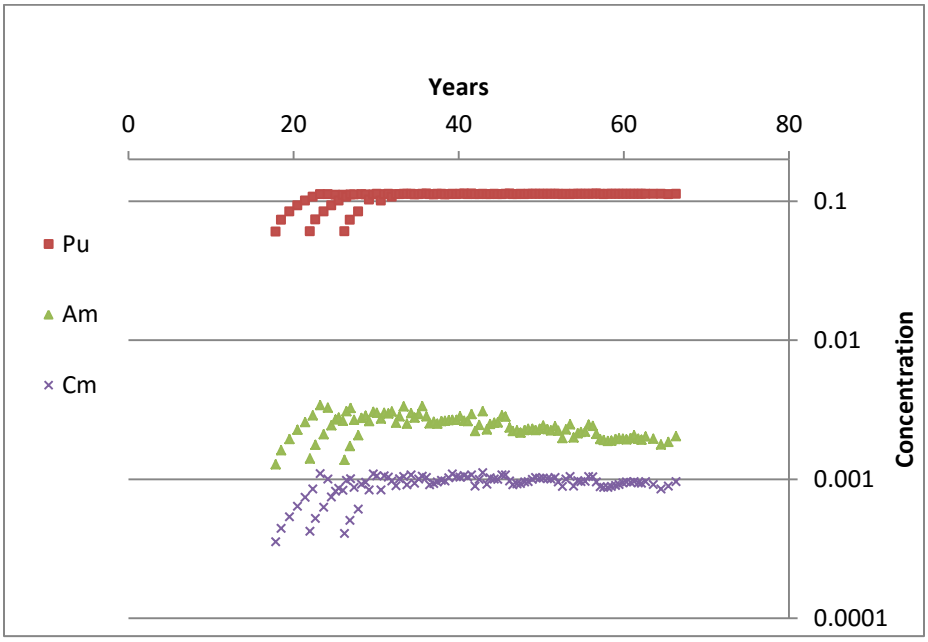

Figure 41 Spent fuel composition for the multi-constraint blending method breeder case.

The compositions of the plutonium vectors for the fresh and spent fuel are seen in Figure 43 and Figure 42. It is possible to see a slight increase in the amount of Pu239 and Pu240 in the fuel between loading new fuel and discharging old fuel. Slight decreases are seen in the other plutonium species. This effect is caused by the blending of the two different types of TRU; one from the LWR fleet and one from the FR fleet. The TRU from the LWR used fuel has a lower fraction of Pu239 than that from the FR. If the TRU for fresh fast reactor fuel was always obtained solely from its own spent fuel one would expect to see the same spent and fresh plutonium composition vectors equal out (radioactive decay was disabled in Cyclus for all of these simulations).

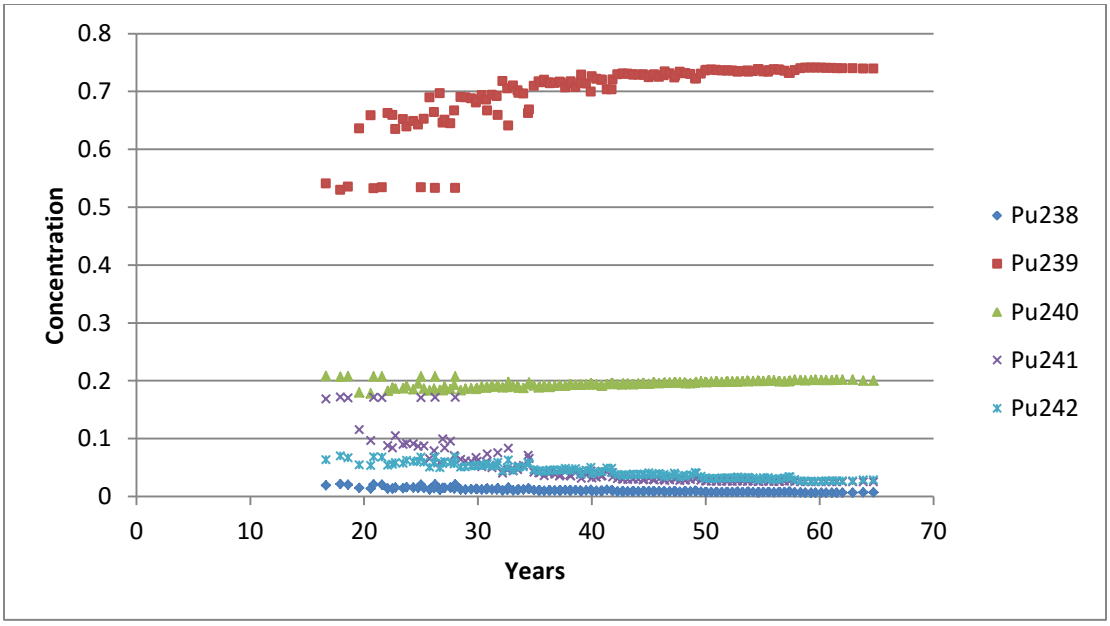

Figure 42 Plutonium isotopic vector in fast reactor fresh fuel for the multi-constraint blending method breeder case

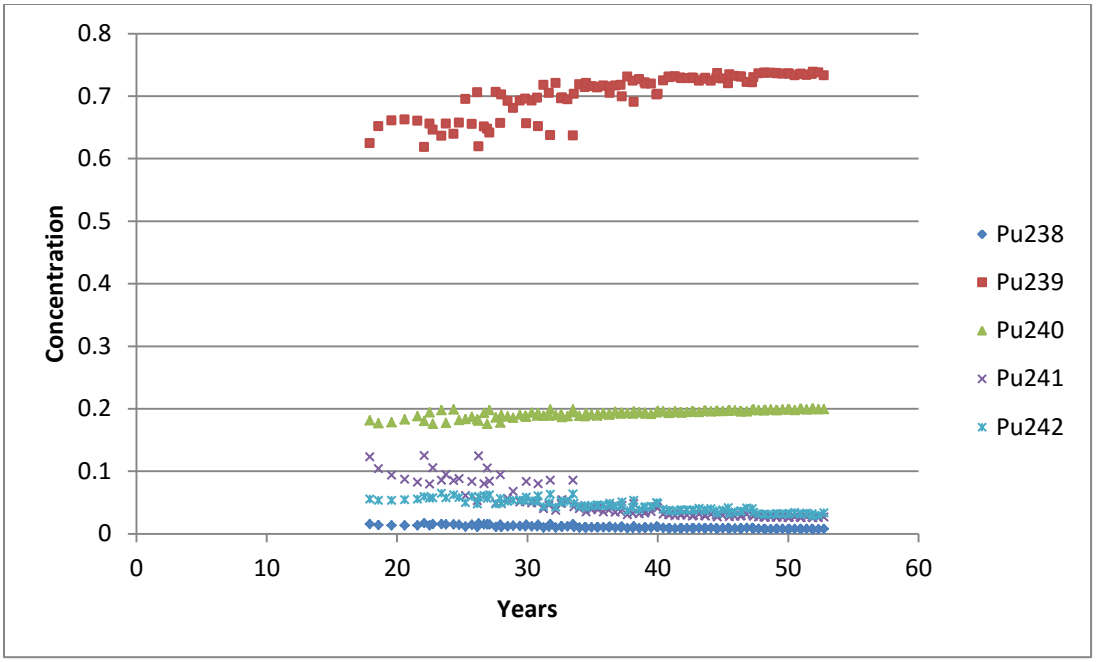

Figure 43 Plutonium isotopic vector in fast reactor spent fuel for the multi-constraint blending method breeder case

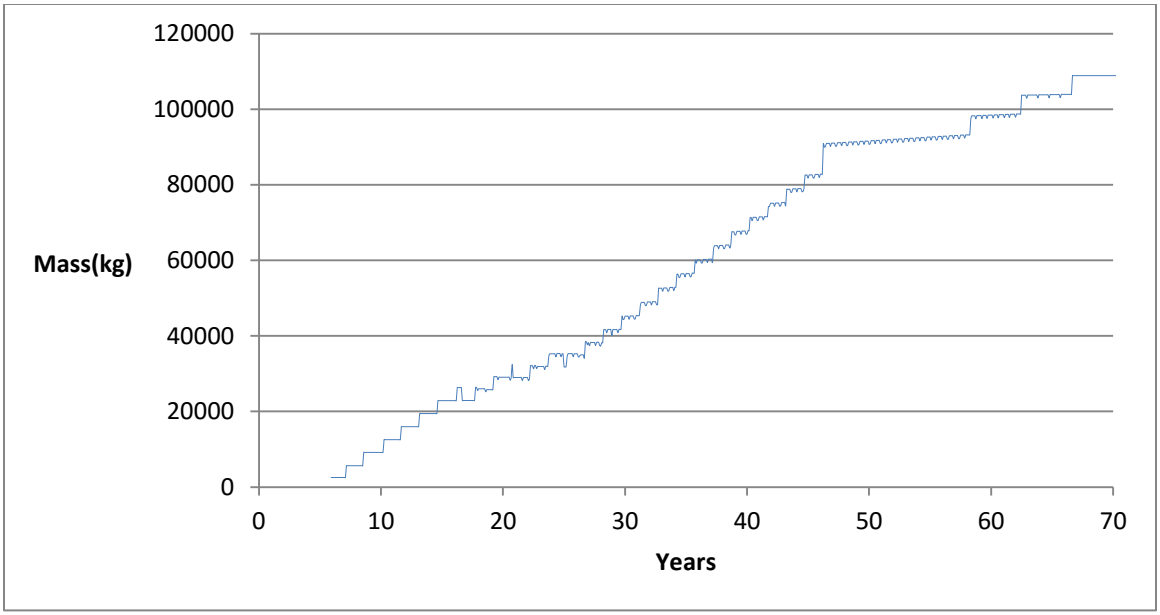

Figure 44 The total free transuranic in the system for the multi-constraint blending method.

The inventory of TRU in the system is seen in Figure 44. As expected for a system featuring FRs with a conversion ratio of greater than one the amount of TRU available is generally increasing. Small dips are withdrawals for FR refueling, while larger dips are withdrawals for FR startup. The large increases at years 45, 58, 63, and 68 come from the shutdown of reactors. The slope of the general increase in TRU inventories is highest while all of the fast reactors and the LWRs are online, since each type of reactor discharges more TRU than it started with, and decreases each time a reactor shuts down.

4.4 Multi-Constraint Blending Sensitivity Analysis

The case studies used to demonstrate the capabilities of Bright-lite's blending methodology show that it is capable of performing these blending tasks for specific reactor parameters. However, it is important to understand how these parameters will

affect the blending methodology, and the results of the fuel cycle in general. To properly test the blending methodology, and Bright-lite requires a parameter sweep over three of the key variables used by Bright-lite in the blending schemes; burnup, conversion ratio, and fluence.

One attribute of the fuel cycle is of particular interest to this study; the power split of the two types of reactor. For example, as the conversion ratio shifts from very low values (0.2) to much higher values (>1.2) the share of each reactor type will shift dramatically. The goal of this is to see what inputs to the fuel cycle, or decisions about its behavior effect the power split the most.

For very low conversion ratios the power split between the LWRs and the FRs will heavily favor the light water reactors. This is because a majority of the fuel for the FRs will come from the LWRs. On the other end of the spectrum for a fuel cycle with high conversion ratio in the FRs, the power split will be much more variable. This is where the blender will be useful in determining how cycles behave.

Based on the constraints that the blender is given the power split between the two reactors may vary dramatically. This is because while the fast reactors are breeding fuel, the blender may decide that the LWR transuranic are better at providing the isotopic composition required for the blend. In this scenario the power split would favor the LWRs still. Whereas if the blender decides that the FR transuranic are more favorable for the desired blend, then the power split will shift toward the fast reactors.

There is currently no logic inside of Bright-lite or Cyclus to go backwards in time and change decisions about the construction of reactors types (FR vs LWR). Therefore, the best technique for testing the effect of changing conversion ratio will be a brute force parameter sweep across conversion ratio, number of fast reactors and number of light water reactors.

This sweep will produce reactor power split vs CR space that represents the possible combination of reactors for a given conversion ratio. It will not provide unique solutions for the power split for a given CR. For example, with a high CR case (1.2) the number of fast reactors is limited by the total amount of TRU being recycled, any number of fast reactors less than that limit will still produce a viable fuel cycle.

Once this space is determined it would be possible to determine which feasible power splits are most desired based on the fuel cycle as a whole using outside criteria. These criteria might the isotopic composition of the material being put into some sort of permanent storage facility.

Additionally, the way in which fuel is reprocessed may also affect the power split of the fuel cycle. Currently, the reprocessing plant reprocesses every time it gets a new batch of spent fuel. It could however, reprocess only when it has a certain amount of fuel stored up and mix all of the spent fuel batches together. Another option would be for the reprocessing plant to mix together spent LWR and FR TRU together while reprocessing.

CONCLUSIONS

The research presented in this work is focused on methods that facilitate analyses of arbitrary or generalized nuclear fuel cycles. In particular, the work aims to enable analyses of new nuclear fuel cycles with unique combinations of technologies or as yet unconceived future technologies. It achieves this by improving the methods for adding and describing new technologies and modeling capabilities in Cyclus as well as enhancing the ability of a Cyclus module, Bright-lite, to calculate material balances for any type of reactor.

The abilities highlighted in CycIC that allow it to adapt to updated or new modules developed in the future are paramount to allowing users to design nuclear fuel cycles graphically. The user experience testing enabled important enhancements to be made to CycIC, notably simplifying the institution view and improving the help system used. Additionally improvements to the save and load system such that it is easy for users to slightly modify their run to test different parameter sets (both inside and outside of CycIC).

The visualization tests help to open up the nuclear fuel cycle to a wider audience by generalizing the method through with the user will interact with the cycle.

The goal of the Bright-lite software is to provide the users with a generic reactor modelling tool that can be easily customized into a variety of different reactor types. The reactor “typing” is handled by the libraries used in Bright-lite. The characterization of the reactor type is done using the powerful tools demonstrated in this work.

The interpolation system is capable of interpolating on the parameters identified for a specific reactor type. This allows Bright-lite to operate with good fidelity using only

a limited number of libraries for that reactor type. This fidelity is dependent on the tolerance that was set during the library generation for that reactor type.

Finally Bright-lite's fuel blending capabilities provide users with a transparent method for blending multiple streams of varying isotopic content into fuel of appropriate composition. The blending capabilities shown in the research demonstrate that Bright-lite is capable of blending fuel to reach burnup or conversion ratio targets as well as other constraints.

While Bright-lite is only a medium fidelity model it provides a good starting point for determining the best way of using the spent fuel from a number of different reactor types inside of a number of different fuel cycle types.

These tools provide for the flexibility and extensibility that is required to endow Cyclus users with the ability to model fuel cycles consisting of a wide range of reactor technologies. The abilities of CycIC allow it to serve as a user interface for any module that may be developed for Cyclus in the future. The techniques used in Bright-lite allow for users to use a single module to simulate a large spectrum of nuclear reactors, both current and future. Together these two pieces of work improve the flexibility of the Cyclus fuel cycle simulator and fuel cycle culture in general by providing users with tools for experimenting with different and unique fuel cycles.

APPENDIX A: USER EXPERIENCE SCENARIOS

User inputs for the first round of user experience testing.

- Source Facility – Front End
    - Outcommodity – Fresh fuel
    - Recipe – Fresh LWR Fuel
    - Capacity – 750kg
    - Other inputs open to user
- Bright-lite Reactor
    - Incommodity – Fresh fuel
    - Outcommodity – Spent fuel
    - Core size – 750kg
    - Batches – 3
    - Burnup - 50
    - Other inputs open to user
- Sink Facility – Back End
    - Incommodity - Spent fuel
    - Capacity – 10,000kg
    - Other inputs open to user

User inputs for the second round of user experience testing.

- Source Facility – U235
    - Outcommodity – U235
    - Recipe – Pure U235
    - Capacity – 100kg
    - Other inputs open to user
- Source Facility – U238
    - Outcommodity – U238
    - Recipe – Pure U238

    - Capacity – 10000kg
    - Other inputs open to user
- Source Facility – Depleted U
    - Outcommodity – DU
    - Recipe – 0.25% enriched U235
    - Capacity – 1000kg
    - Other inputs open to user
- Bright-lite UOX Fuel Fab
    - Nonfissle Stream – U238
    - Fissle stream – U235
    - Outcommodity – Fresh Fuel
    - Other inputs open to user
- Bright-lite MOX Fuel Fab
    - Nonfissle Stream – DU
    - Fissle stream – LWR TRU
    - Outcommodity – Fresh MOX Fuel
    - Other inputs open to user
- Bright-lite Reprocessing Plant
    - Incommodity – Spent LWR Fuel
    - Outcommodity – LWR TRU / Waste
    - Other inputs open to user
- Bright-lite Reactor - UOX
    - Incommodity – Fresh fuel
    - Outcommodity – Spent LWR fuel
    - Core size – 750kg
    - Batches – 3
    - Burnup - 50
    - Other inputs open to user
- Bright-lite Reactor - MOX

    - Incommodity – Fresh MOX fuel
    - Outcommodity – waste
    - Core size – 750kg
    - Batches – 4
    - Burnup - 50
    - Other inputs open to user
- Sink Facility – Back End
    - Incommodity - Spent fuel
    - Capacity – 10,000kg
    - Other inputs open to user
- Deploy Institution
    - Initial facilities
        - U235 – 1
        - U238 – 1
        - Depleted U – 1
        - UOX Fuel Fab – 1
        - MOX Fuel Fab – 1
        - Reprocessing Plant – 1
    - Deployed Facilities
        - UOX Reactor
            - Number: 4
            - Date: 10
        - UOX Reactor
            - Number: 4
            - Date: 30
        - MOX Reactor
            - Number: 1
            - Date: 100
- Null Region

APPENDIX B: CYCLUS SAMPLE INPUTS

Criticality Blending

2 Source, 1 FuelFab 1 LWR, 1 Sink
Reactor database: extLWR
Fuel: Calculated
Batches: 3

"time cyclus examples/LWRblending.xml" = real 0m3.676s (2015/02/26)
real 0m3.407s (2015/03/04)

-->

```
<simulation>
 <control>
  <duration>600</duration>
  <startmonth>1</startmonth>
  <startyear>2000</startyear>
 </control>

 <archetypes>
  <spec><lib>agents</lib><name>Source</name></spec>
  <spec><lib>Brightlite</lib><name>ReactorFacility</name></spec>
  <spec><lib>Brightlite</lib><name>FuelfabFacility</name></spec>
  <spec><lib>agents</lib><name>Sink</name></spec>
  <spec><lib>agents</lib><name>NullRegion</name></spec>
  <spec><lib>agents</lib><name>NullInst</name></spec>
  <spec><lib>cycamore</lib><name>DeployInst</name></spec>
 </archetypes>

 <facility>
  <name>Fissile_Source</name>
  <config>
   <Source>
    <commod>fissile_stream</commod>
    <recipe_name>U235</recipe_name>
          <capacity>400000.0</capacity>
   </Source>
  </config>
 </facility>

 <facility>
  <name>NFissile_Source</name>
  <config>
   <Source>
    <commod>nfissile_stream</commod>
    <recipe_name>U238</recipe_name>
          <capacity>200000.0</capacity>
   </Source>
  </config>
 </facility>

 <facility>
  <name>Sink</name>
  <config>
   <Sink>
    <in_commods>
     <val>uf</val>
    </in_commods>
    <capacity>1000000.0</capacity>
   </Sink>
```

```
  </config>
</facility>

<facility>
  <name>Rx</name>
  <config>
    <ReactorFacility>
      <in_commods>
        <val>UOX</val>
      </in_commods>
      <out_commod>uf</out_commod>
      <libraries><val>extLWR</val></libraries>
      <target_burnup>50</target_burnup>
      <nonleakage>0.883</nonleakage>
      <core_mass>127000.0</core_mass>
      <generated_power>4212</generated_power>
      <batches>4</batches>
      <outage_time>30</outage_time>
      <flux_mode>0</flux_mode>
      <tolerance>0.0001</tolerance>
      <SS_tolerance>0.0001</SS_tolerance>
      <burnupcalc_timestep>60</burnupcalc_timestep>
      <CR_fissile>
        <val>922350</val>
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>

        <facility>
                <name>FF_UOX</name>
                <config>
                        <FuelfabFacility>
                        <fissle_stream>fissile_stream</fissle_stream>
                        <non_fissle_stream>nfissile_stream</non_fissle_stream>
                        <in_commods>
                                <item><key>blank</key><val>0.0</val></item>
                        </in_commods>
                        <out_commod>UOX</out_commod>
                </FuelfabFacility></config>
        </facility>

<region>

  <name>SingleRegion</name>
  <config> <NullRegion/> </config>
  <institution>
    <name>SingleInstitution</name>
    <initialfacilitylist>
      <entry>
        <prototype>Fissile_Source</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>NFissile_Source</prototype>
        <number>1</number>
      </entry>
```

```
    <entry>
      <prototype>FF_UOX</prototype>
      <number>1</number>
    </entry>
    <entry>
      <prototype>Sink</prototype>
      <number>1</number>
    </entry>
   </initialfacilitylist>
   <config> <NullInst/> </config>
  </institution>
        <institution>
                <name>utility</name>
                <config>
                <DeployInst>
                        <prototypes><val>Rx</val></prototypes>
                        <build_times><val>50</val></build_times>
                        <n_build><val>1</val></n_build>
                </DeployInst>
                </config>
        </institution>
 </region>

 <recipe>
  <name>U235</name>
  <basis>mass</basis>
  <nuclide>
   <id>U235</id>
   <comp>1</comp>
  </nuclide>
 </recipe>

 <recipe>
  <name>U238</name>
  <basis>mass</basis>
  <nuclide>
   <id>U238</id>
   <comp>1</comp>
  </nuclide>
```

CR Blending

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
        <simulation><control><duration>2500</duration><startmonth>1</startmonth><startyear>0</startyear></control>
<archetypes><spec><lib>Brightlite</lib><name>FuelfabFacility</name></spec><spec><lib>agents</lib><name>Source</name></s
pec><spec><lib>agents</lib><name>Source</name></spec><spec><lib>Brightlite</lib><name>FuelfabFacility</name></spec><sp
ec><lib>Brightlite</lib><name>ReprocessFacility</name></spec><spec><lib>Brightlite</lib><name>ReactorFacility</name></spe
c><spec><lib>Brightlite</lib><name>ReactorFacility</name></spec><spec><lib>Brightlite</lib><name>ReprocessFacility</name
></spec><spec><lib>agents</lib><name>Sink</name></spec><spec><lib>agents</lib><name>NullRegion</name></spec><spec>
<lib>agents</lib><name>NullInst</name></spec><spec><lib>cycamore</lib><name>DeployInst</name></spec></archetypes>

<commodity><name>LWR Fuel FAb</name><solution_priority>1.0</solution_priority></commodity><commodity><name>LWR
Spent Fuel</name><solution_priority>1.0</solution_priority></commodity><commodity><name>FR Spent
Fuel</name><solution_priority>1.0</solution_priority></commodity><commodity><name>FR
Fuel</name><solution_priority>1.0</solution_priority></commodity><commodity><name>DU</name><solution_priority>1.0</sol
ution_priority></commodity><commodity><name>LWR
Reprocessed</name><solution_priority>1.0</solution_priority></commodity><commodity><name>FR
Reprocessed</name><solution_priority>1.0</solution_priority></commodity><commodity><name>U235</name><solution_priority
>1.0</solution_priority></commodity><commodity><name>U238</name><solution_priority>1.0</solution_priority></commodity>
<commodity><name>FR
Fuel</name><solution_priority>1.0</solution_priority></commodity><commodity><name>WASTE</name><solution_priority>1.0
</solution_priority></commodity>

<facility>
```

```
        <name>FR Fuel Fab</name>
        <config>
                <FuelfabFacility>
                        <maximum_storage>9.9999999999999999E+59</maximum_storage>
                        <fissle_stream>LWR Reprocessed</fissle_stream>
                        <non_fissle_stream>DU2</non_fissle_stream>
                        <in_commods>
                                <item><key>FR Reprocessed</key><val>1.0</val></item>
                        </in_commods>
                        <out_commod>FR Fuel</out_commod>
                </FuelfabFacility>
        </config>
</facility>

<facility>
        <name>MineU235</name>
        <config>
                <Source>
                        <commod>U235</commod>
                        <recipe_name>U235</recipe_name>
                        <capacity>40000</capacity>
                </Source>
        </config>
</facility>

<facility>
        <name>U238</name>
        <config>
                <Source>
                        <commod>U238</commod>
                        <recipe_name>Uranium 238</recipe_name>
                        <capacity>200000</capacity>
                </Source>
        </config>
</facility>

<facility>
        <name>DU</name>
        <config><Source>
                        <commod>DU</commod>
                        <recipe_name>DU</recipe_name>
                        <capacity>20000</capacity>
        </Source></config>
</facility>

<facility>
        <name>DU2</name>
        <config><Source>
                        <commod>DU2</commod>
                        <recipe_name>DU</recipe_name>
                        <capacity>20000</capacity>
        </Source></config>
</facility>

<facility>
        <name>LWR Fuel FAb</name>
        <config><FuelfabFacility>
                <maximum_storage>9.9999999999999999E+59</maximum_storage>
                <fissle_stream>U235</fissle_stream>
                <non_fissle_stream>U238</non_fissle_stream>
                <in_commods><item><key>DU</key><val>0.0</val></item></in_commods>
                <out_commod>LWR Fuel</out_commod>
        </FuelfabFacility></config>
</facility>
```

```
<facility>
        <name>LWR Seperation</name>
        <config><ReprocessFacility>
                <in_commod><val>uf</val></in_commod>
                <commod_out>
                        <val>LWR Reprocessed</val>
                        <val>WASTE</val>
                </commod_out>
                <repro_input_path>input/FR_reprocess.txt</repro_input_path>
                <max_inv_size>1.000000000000000E+299</max_inv_size>
                <input_capacity>400000</input_capacity>
                <output_capacity>400000</output_capacity>
        </ReprocessFacility></config>
</facility>

 <facility>
  <name>LWR</name>
  <config>
   <ReactorFacility>
    <in_commods>
      <val>LWR Fuel</val>
    </in_commods>
    <out_commod>uf</out_commod>
    <libraries><val>extLWR</val></libraries>
    <target_burnup>50.0</target_burnup>
    <nonleakage>0.973</nonleakage>
    <core_mass>140000.0</core_mass>
    <generated_power>4000.0</generated_power>
    <batches>4</batches>
    <outage_time>30</outage_time>
    <flux_mode>2</flux_mode>
    <tolerance>0.001</tolerance>
    <SS_tolerance>0.0001</SS_tolerance>
    <burnupcalc_timestep>60</burnupcalc_timestep>
    <CR_fissile>
      <val>922350</val>
      <val>942380</val>
      <val>942390</val>
      <val>942400</val>
      <val>942410</val>
      <val>942420</val>
      <val>952400</val>
      <val>952420</val>
      <val>952440</val>
    </CR_fissile>
   </ReactorFacility>
  </config>
 </facility>

        <facility>
                <name>FRx</name>
                <config>
                <ReactorFacility>
                <tolerance>0.0010</tolerance>
                <out_commod>FR Spent Fuel</out_commod>
                <libraries><val>FR50</val></libraries>
                <batches>6</batches>
                <nonleakage>0.58</nonleakage>
                <target_burnup>52.62</target_burnup>
                <generated_power>2000.0</generated_power>
                <core_mass>45000</core_mass>
                <burnupcalc_timestep>100</burnupcalc_timestep>
                <flux_mode>0</flux_mode>
                <reactor_life>500</reactor_life>
                <CR_fissile>
```

```
                        <val>942380</val>
                        <val>942390</val>
                        <val>942400</val>
                        <val>942410</val>
                        <val>942420</val>
                        <val>952410</val>
                        <val>952430</val>
                        <val>962420</val>
                        <val>942440</val>
                </CR_fissile>
        <CR_target>1.2</CR_target>
        <in_commods>
           <val>FR Fuel</val>
        </in_commods>
</ReactorFacility></config></facility>

<facility>
                <name>FR Reprocess</name>
                <config><ReprocessFacility>
                        <in_commod>
                                <val>FR Spent Fuel</val>
                        </in_commod>
                        <commod_out>
                                <val>FR Reprocessed</val>
                                <val>WASTE</val>
                        </commod_out>
                        <repro_input_path>input/FR_reprocess.txt</repro_input_path>
                        <max_inv_size>1.000000000000000E+299</max_inv_size>
                        <input_capacity>20000000</input_capacity>
                        <output_capacity>2000000</output_capacity>
                </ReprocessFacility></config>
</facility>

<facility>
                <name>SINK</name>
                <config><Sink>
                        <in_commods>
                                <val>WASTE</val>
                        </in_commods>
                        <capacity>100000</capacity>
                        <max_inv_size>1.000000000000000E+299</max_inv_size>
                </Sink></config>
</facility>

<region><name>USA</name><config><NullRegion/></config><institution><initialfacilitylist><entry><prototype>MineU235</prot
otype><number>1</number></entry><entry><prototype>U238</prototype><number>1</number></entry><entry><prototype>DU<
/prototype><number>1</number></entry><entry><prototype>DU2</prototype><number>1</number></entry><entry><prototype>L
WR Fuel FAb</prototype><number>1</number></entry><entry><prototype>LWR
Seperation</prototype><number>1</number></entry><entry><prototype>FR
Reprocess</prototype><number>1</number></entry><entry><prototype>SINK</prototype><number>1</number></entry><entry>
<prototype>FR Fuel
Fab</prototype><number>1</number></entry></initialfacilitylist><name>utility</name><config><NullInst/></config></institution
><institution><name>utility2</name><config><DeployInst><prototypes><val>LWR</val><val>FRx</val></prototypes><build_ti
mes><val>50</val><val>300</val></build_times><n_build><val>2</val><val>1</val></n_build></DeployInst></config></institut
ion></region>

<recipe><name>Uranium
238</name><basis>mass</basis><nuclide><id>922380</id><comp>100.0000002</comp></nuclide></recipe><recipe><name>U23
5</name><basis>mass</basis><nuclide><id>922350</id><comp>100.0000002</comp></nuclide></recipe><recipe><name>DU</n
ame><basis>mass</basis><nuclide><id>922350</id><comp>0.2500002</comp></nuclide><nuclide><id>922380</id><comp>99.7
500002</comp></nuclide></recipe></simulation>
```

Transition Study: Burner

```
<!-- uox source, 8 LWR, reprocess, mox reactor, sinks -->

<simulation>
 <control>
  <duration>600</duration>
  <startmonth>1</startmonth>
  <startyear>2000</startyear>
 </control>

 <archetypes>
  <spec><lib>agents</lib><name>Source</name></spec>
  <spec><lib>Brightlite</lib><name>ReactorFacility</name></spec>
  <spec><lib>Brightlite</lib><name>FuelfabFacility</name></spec>
  <spec><lib>Brightlite</lib><name>ReprocessFacility</name></spec>
  <spec><lib>agents</lib><name>Sink</name></spec>
  <spec><lib>agents</lib><name>NullRegion</name></spec>
  <spec><lib>cycamore</lib><name>DeployInst</name></spec>
 </archetypes>

 <facility>
  <name>Source</name>
  <config>
   <Source>
    <commod>fbl</commod>
    <recipe_name>U235</recipe_name>
          <capacity>100.0</capacity>
   </Source>
  </config>
 </facility>

 <facility>
  <name>Source1</name>
  <config>
   <Source>
    <commod>fbl1</commod>
    <recipe_name>U238</recipe_name>
            <capacity>4450.0</capacity>
   </Source>
  </config>
 </facility>

 <facility>
  <name>Source2</name>
  <config>
   <Source>
    <commod>NU</commod>
    <recipe_name>NU_recipe</recipe_name>
            <capacity>100.0</capacity>
   </Source>
  </config>
 </facility>

 <facility>
  <name>Sink2</name>
  <config>
   <Sink>
    <in_commods>
     <val>reproFuel1</val>
    </in_commods>
    <capacity>1000000.0</capacity>
   </Sink>
  </config>
```

```
</facility>

<facility>
  <name>Sink3</name>
  <config>
    <Sink>
      <in_commods>
        <val>spentFF</val>
      </in_commods>
      <capacity>1000000.0</capacity>
    </Sink>
  </config>
</facility>

<facility>
  <name>Rx1</name>
  <config>
    <ReactorFacility>
      <in_commods>
          <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>extLWR</val></libraries>
      <target_burnup>35.0</target_burnup>
      <nonleakage>0.966</nonleakage>
      <core_mass>1</core_mass>
      <batches>3</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
          <val>902250</val>
          <val>902270</val>
          <val>902290</val>
          <val>922350</val>
          <val>942380</val>
          <val>942390</val>
          <val>942400</val>
          <val>942410</val>
          <val>942420</val>
          <val>952400</val>
          <val>952420</val>
          <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>

<facility>
  <name>Rx2</name>
  <config>
    <ReactorFacility>
      <in_commods>
          <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>standLWR</val></libraries>
      <target_burnup>40.0</target_burnup>
      <nonleakage>0.948</nonleakage>
      <core_mass>1</core_mass>
      <batches>3</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
```

```
        <val>902250</val>
        <val>902270</val>
        <val>902290</val>
        <val>922350</val>
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>


<facility>
  <name>Rx3</name>
  <config>
    <ReactorFacility>
      <in_commods>
        <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>extLWR</val></libraries>
      <target_burnup>45.0</target_burnup>
      <nonleakage>0.966</nonleakage>
      <core_mass>1</core_mass>
      <batches>4</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
        <val>902250</val>
        <val>902270</val>
        <val>902290</val>
        <val>922350</val>
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>

<facility>
  <name>FFUOX</name>
  <config>
    <FuelfabFacility>
     <out_commod>fuel</out_commod><fissle_stream>fbl</fissle_stream><non_fissle_stream>fbl1</non_fissle_stream>
    </FuelfabFacility>
  </config>
</facility>

<facility>
  <name>Repro</name>
  <config>
    <ReprocessFacility>
```

```
    <in_commod>
      <val>spentFuel</val>
    </in_commod>
    <commod_out>
      <val>reproFuel0</val>
      <val>reproFuel1</val>
    </commod_out>
    <repro_input_path>../Bright-lite/input/reprocess_input.txt</repro_input_path>
    <input_capacity>1000000</input_capacity>
    <output_capacity>1000000</output_capacity>
   </ReprocessFacility>
  </config>
</facility>

<facility>
  <name>FFFR</name>
  <config>
    <FuelfabFacility>
 <out_commod>FFfuel</out_commod><fissle_stream>reproFuel0</fissle_stream><non_fissle_stream>NU</non_fissle_stream>
    </FuelfabFacility>
  </config>
</facility>

<facility>
  <name>RxFR</name>
  <config>
    <ReactorFacility>
     <in_commods>
       <val>FFfuel</val>
     </in_commods>
     <out_commod>spentFF</out_commod>
     <libraries><val>FR50</val></libraries>
     <target_burnup>185.0</target_burnup>
     <CR_target>0.8</CR_target>
     <nonleakage>0.57</nonleakage>
     <core_mass>300.0</core_mass>
           <interpol_pairs>
       <key>BURNUP</key>
              <val>185</val>
     </interpol_pairs>
     <batches>6</batches>
     <flux_mode>2</flux_mode>
     <cylindrical_delta>10</cylindrical_delta>
     <burnupcalc_timestep>40</burnupcalc_timestep>
     <CR_fissile>
       <val>902250</val>
       <val>902270</val>
       <val>902290</val>
       <val>922350</val>
       <val>942380</val>
       <val>942390</val>
       <val>942400</val>
       <val>942410</val>
       <val>942420</val>
       <val>952400</val>
       <val>952420</val>
       <val>952440</val>
     </CR_fissile>
   </ReactorFacility>
  </config>
</facility>

<region>
  <name>SingleRegion</name>
  <config> <NullRegion/> </config>
```

```
 <institution>
  <name>SingleInstitution</name>
  <initialfacilitylist>
   <entry>
    <prototype>Source</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Source1</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Source2</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Rx1</prototype>
    <number>2</number>
   </entry>
   <entry>
    <prototype>FFUOX</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Sink2</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Sink3</prototype>
    <number>1</number>
   </entry>
   <entry>
    <prototype>Repro</prototype>
    <number>1</number>
   </entry>
  </initialfacilitylist>
  <config>
 <DeployInst>
   <buildorder>
    <prototype>RxFR</prototype>
      <number>1</number>
      <date>200</date>
   </buildorder>
   <buildorder>
    <prototype>Rx2</prototype>
      <number>2</number>
      <date>60</date>
   </buildorder>
   <buildorder>
    <prototype>Rx3</prototype>
      <number>2</number>
      <date>120</date>
   </buildorder>
   <buildorder>
    <prototype>FFMOX</prototype>
      <number>1</number>
      <date>8</date>
   </buildorder>
 </DeployInst>
  </config>
 </institution>
</region>

<recipe>
 <name>U235</name>
```

```
  <basis>mass</basis>
  <nuclide>
    <id>922350000</id>
    <comp>100</comp>
  </nuclide>
 </recipe>

 <recipe>
  <name>U238</name>
  <basis>mass</basis>
  <nuclide>
    <id>922380000</id>
    <comp>100</comp>
  </nuclide>
 </recipe>

 <recipe>
  <name>NU_recipe</name>
  <basis>mass</basis>
  <nuclide>
    <id>922380000</id>
    <comp>99.289</comp>
  </nuclide>
  <nuclide>
    <id>922350000</id>
    <comp>0.711</comp>
  </nuclide>
 </recipe>
</simulation>
```

Transition Study: Breeder

```
<!-- uox source, 8 LWR, reprocess, mox reactor, sinks -->

<simulation>
 <control>
  <duration>600</duration>
  <startmonth>1</startmonth>
  <startyear>2000</startyear>
 </control>

 <archetypes>
  <spec><lib>agents</lib><name>Source</name></spec>
  <spec><lib>Brightlite</lib><name>ReactorFacility</name></spec>
  <spec><lib>Brightlite</lib><name>FuelfabFacility</name></spec>
  <spec><lib>Brightlite</lib><name>ReprocessFacility</name></spec>
  <spec><lib>agents</lib><name>Sink</name></spec>
  <spec><lib>agents</lib><name>NullRegion</name></spec>
  <spec><lib>cycamore</lib><name>DeployInst</name></spec>
 </archetypes>

 <facility>
  <name>Source</name>
  <config>
    <Source>
      <commod>fbl</commod>
      <recipe_name>U235</recipe_name>
            <capacity>100.0</capacity>
    </Source>
  </config>
 </facility>

 <facility>
  <name>Source1</name>
```

```
  <config>
    <Source>
      <commod>fbl1</commod>
      <recipe_name>U238</recipe_name>
                <capacity>4450.0</capacity>
    </Source>
  </config>
</facility>

<facility>
  <name>Source2</name>
  <config>
    <Source>
      <commod>NU</commod>
      <recipe_name>NU_recipe</recipe_name>
                <capacity>100.0</capacity>
    </Source>
  </config>
</facility>

<facility>
  <name>Sink2</name>
  <config>
    <Sink>
      <in_commods>
        <val>reproFuel1</val>
      </in_commods>
      <capacity>1000000.0</capacity>
    </Sink>
  </config>
</facility>

<facility>
  <name>Sink3</name>
  <config>
    <Sink>
      <in_commods>
        <val>spentFF</val>
      </in_commods>
      <capacity>1000000.0</capacity>
    </Sink>
  </config>
</facility>

<facility>
  <name>Rx1</name>
  <config>
    <ReactorFacility>
      <in_commods>
          <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>extLWR</val></libraries>
      <target_burnup>35.0</target_burnup>
      <nonleakage>0.966</nonleakage>
      <core_mass>1</core_mass>
      <batches>3</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
        <val>902250</val>
        <val>902270</val>
        <val>902290</val>
        <val>922350</val>
```

```
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>

<facility>
  <name>Rx2</name>
  <config>
    <ReactorFacility>
      <in_commods>
        <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>standLWR</val></libraries>
      <target_burnup>40.0</target_burnup>
      <nonleakage>0.948</nonleakage>
      <core_mass>1</core_mass>
      <batches>3</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
        <val>902250</val>
        <val>902270</val>
        <val>902290</val>
        <val>922350</val>
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
      </CR_fissile>
    </ReactorFacility>
  </config>
</facility>


<facility>
  <name>Rx3</name>
  <config>
    <ReactorFacility>
      <in_commods>
        <val>fuel</val>
      </in_commods>
      <out_commod>spentFuel</out_commod>
      <libraries><val>extLWR</val></libraries>
      <target_burnup>45.0</target_burnup>
      <nonleakage>0.966</nonleakage>
      <core_mass>1</core_mass>
      <batches>4</batches>
      <flux_mode>1</flux_mode>
      <cylindrical_delta>10</cylindrical_delta>
      <burnupcalc_timestep>40</burnupcalc_timestep>
      <CR_fissile>
```

```
        <val>902250</val>
        <val>902270</val>
        <val>902290</val>
        <val>922350</val>
        <val>942380</val>
        <val>942390</val>
        <val>942400</val>
        <val>942410</val>
        <val>942420</val>
        <val>952400</val>
        <val>952420</val>
        <val>952440</val>
     </CR_fissile>
   </ReactorFacility>
  </config>
</facility>

<facility>
  <name>FFUOX</name>
  <config>
    <FuelfabFacility>
     <out_commod>fuel</out_commod><fissle_stream>fbl</fissle_stream><non_fissle_stream>fbl1</non_fissle_stream>
    </FuelfabFacility>
  </config>
</facility>

<facility>
  <name>Repro</name>
  <config>
    <ReprocessFacility>
      <in_commod>
         <val>spentFuel</val>
      </in_commod>
      <commod_out>
         <val>reproFuel0</val>
         <val>reproFuel1</val>
      </commod_out>
      <repro_input_path>../Bright-lite/input/reprocess_input.txt</repro_input_path>
      <input_capacity>1000000</input_capacity>
      <output_capacity>1000000</output_capacity>
    </ReprocessFacility>
  </config>
</facility>

<facility>
  <name>FFFR</name>
  <config>
    <FuelfabFacility>
 <out_commod>FFfuel</out_commod><fissle_stream>reproFuel0</fissle_stream><non_fissle_stream>NU</non_fissle_stream>
    </FuelfabFacility>
  </config>
</facility>

<facility>
  <name>RxFR</name>
  <config>
    <ReactorFacility>
      <in_commods>
         <val>FFfuel</val>
      </in_commods>
      <out_commod>spentFF</out_commod>
      <libraries><val>FR50</val></libraries>
      <target_burnup>185.0</target_burnup>
      <CR_target>1.2</CR_target>
     <fluence_limit>3.3E24</fluence_limit>
```

```
    <nonleakage>0.57</nonleakage>
    <core_mass>300.0</core_mass>
          <interpol_pairs>
      <key>BURNUP</key>
              <val>185</val>
    </interpol_pairs>
    <batches>6</batches>
    <flux_mode>2</flux_mode>
    <cylindrical_delta>10</cylindrical_delta>
    <burnupcalc_timestep>40</burnupcalc_timestep>
    <CR_fissile>
      <val>902250</val>
      <val>902270</val>
      <val>902290</val>
      <val>922350</val>
      <val>942380</val>
      <val>942390</val>
      <val>942400</val>
      <val>942410</val>
      <val>942420</val>
      <val>952400</val>
      <val>952420</val>
      <val>952440</val>
    </CR_fissile>
   </ReactorFacility>
  </config>
</facility>

<region>
  <name>SingleRegion</name>
  <config> <NullRegion/> </config>
  <institution>
    <name>SingleInstitution</name>
    <initialfacilitylist>
      <entry>
        <prototype>Source</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Source1</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Source2</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Rx1</prototype>
        <number>2</number>
      </entry>
      <entry>
        <prototype>FFUOX</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Sink2</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Sink3</prototype>
        <number>1</number>
      </entry>
      <entry>
        <prototype>Repro</prototype>
        <number>1</number>
```

```
      </entry>
    </initialfacilitylist>
    <config>
  <DeployInst>
    <buildorder>
      <prototype>RxFR</prototype>
        <number>1</number>
        <date>200</date>
    </buildorder>
    <buildorder>
      <prototype>Rx2</prototype>
        <number>2</number>
        <date>60</date>
    </buildorder>
    <buildorder>
      <prototype>Rx3</prototype>
        <number>2</number>
        <date>120</date>
    </buildorder>
    <buildorder>
      <prototype>FFMOX</prototype>
        <number>1</number>
        <date>8</date>
    </buildorder>
  </DeployInst>
    </config>
  </institution>
</region>

<recipe>
  <name>U235</name>
  <basis>mass</basis>
  <nuclide>
    <id>922350000</id>
    <comp>100</comp>
  </nuclide>
</recipe>

<recipe>
  <name>U238</name>
  <basis>mass</basis>
  <nuclide>
    <id>922380000</id>
    <comp>100</comp>
  </nuclide>
</recipe>

<recipe>
  <name>NU_recipe</name>
  <basis>mass</basis>
  <nuclide>
    <id>922380000</id>
    <comp>99.289</comp>
  </nuclide>
  <nuclide>
    <id>922350000</id>
    <comp>0.711</comp>
  </nuclide>
</recipe>
</simulation>
```

REFERENCES

[1] R. Gregg, "ORION 'The tip of the fuel cycle modelling iceburg"," National Nuclear Laboratory, April 2014. [Online]. Available: http://www.nnl.co.uk/media/1581/nnl-tech-conference-poster-rg-orion-apr-14.pdf. [Accessed 25 August 2014].

[2] R. Gregg and K. Hesketh, Writers, *ORION and its application to fuel cycle assessment in the UK.* [Performance]. National Nuclear Laboratory, 2014.

[3] J. J. Jacobson, G. E. Matthern, S. J. Piet and D. E. Shropshire, "VISION: Verifiable Fuel Cycle Simulation Model," in *ANFM*, 2009.

[4] M. Reno and e. all, *System Dynamics Modeling using Powersim Studio,* Sandia Nation Laboratories, 2007.

[5] Z. Mijailović and D. Milićev, "Empirical analysis of GUI programming concerns," *International Journal of Human-Computer Studies,* vol. 72, no. 10-11, pp. 757-771, 2014.

[6] H. Lam and e. al, "Emperical Studies in Information Visualization: Seven Scenarios," *IEEE Transactions on Visualization and Computer Graphsics,* vol. 18, no. 9, pp. 1520-1537, 2012.

[7] S. B and P. C, "Strategies for Evaluating Information Visualization Tools: Multi-dimensional In-depth Long-term Case Studies," in *Advanced Visual Interfaces Conference Proceedings*, Venice, Italy, 2006.

[8] S. J and S. B, "Knowledge Discovery in High-Dimensional Data: Case Studies and a User Survey for the Rank-by-Feature Framework," *IEEE Transactions on Visualization and Computer Graphics,* vol. 12, no. 3, pp. 311-323, 2006.

[9] K. Oliver, *GeniusV2: Software Design and Methematical Formulations for Multi-region Discrete Nuclear Fuel Cycle Simulation and Analysis,* Madison, 2009.

[10] E. A. Schneider, C. G. Bathke and M. R. James, "NFCSIM: A Dynamic Fuel Burnup and Fuel Cycle Simulation Tool," *Nuclear Technology,* vol. 151, 2005.

[11] S. E, Interviewee, *Proposal Review: Literature Reivew.* [Interview]. August 2014.

[12] L. Guerin, "Impact of Alternative Nuclear Fuel Cycle Options on Infrastructure and Fuel Requirements, Actinide and Waste Inventories, and Economics.," MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2009.

[13] J. Y. B. B. e. a. F. Daniel, "Understanding UI Integration: A survey of problems, technologies, and opportunities," *IEEE Internet Computing,* vol. 11, pp. 59-66, 2007.

[14] MIT, "OpenMC User's Guide," MIT, 2014. [Online]. Available: http://mit-crpg.github.io/openmc/usersguide/index.html. [Accessed August 2014].

[15] J. Leppanen, *Serpent - a Continuous Energy Monte Carlo Reactor Physics Burnup Calculation Code,* VTT Technical Research Centre of Finland, 2013.

[16] R. Flanagan, C. Bagdatlioglu and E. Schneider, "Fuel Composition Techniques of Nuclear Fuel Cycle Simulators," in *Physor 2014 - The Role of Reactor Physics*

Toward a Sustainable Future, Kyoto, Japan, 2014.

[17] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *ACM Nat. Conf.*, 1968.

[18] W. Gorgon and J. Wixom, "Shepard's Method of "Metric Interpolation" to Bivariate and Multivariate Interpolation," *Mathematics of Computation Volume 32, Number 141,* pp. 253-264, 1978.

[19] J. X. A. Scopatz, "XSGen," [Online]. Available: https://github.com/scopatz/xsgen. [Accessed 8 8 2015].

[20] O. R. N. Laboratory, *Origen 2, Version 2.2,* Oak Ridge, Tennessee, June 2002.

[21] B. J. C. Barthold W.P., "Performance Characteristics of Homogeneous Versus Heterogeneous Liquid-Metal Fast Breeder Reactors," Argonne National Laboratory, Applied Physics Divisions, Argonne, Illinois, 1978.

[22] Y. U. H. Sekimoto, "Effect of Fuel and Coolant Temperatures and Nuetron Fluence on CANDLE Burnup Calculation," Journal of Nuclear Science and Technology, 2006.

[23] M. Tory, D. Sprague, F. Wu, W. So and T. Munzner, "Spatial Spatialization Design: Comparing Points and Landscapes," 2007.

[24] M. Gidden, "An Agent-Based Modeling Framework and Application for the Generic Nuclear Fuel Cycle," 2013.

[25] R. Sanchez, I. Zmijarevic, M. Coste, E. Masielo, S. Santandrea, E. Martinolli, L. Villate, N. Schwartz and N. Guler, "APOLLO2 YEAR 2010," 2010.

[26] J. Grouiller, J. Vidal, A. Launay, Y. Berthion, A. Marc and H. Toubon, "CESAR: A Code for Nuclear Fuel and Waste Characterisation," in *WM*, Tucson, AZ, 2006.

[27] R. Gregg, Writer, *Workshop on Evaluation of Waste Streams Associated with LWR Fuel Cycle Options.* [Performance]. National Nuclear Laboratory, 2011.

[28] Cyclus, "Cyclus Fuel Cycle Simulation," 2013. [Online]. Available: www.fuelcycle.org. [Accessed Feburary 2014].

[29] Y. O. S. K. Y. Ishiwatari, "Breeding Ratio Analysis of a Fast Reactor Cooled by Supercritical Light Water," Journal of Nuclear Science and Technology, 2012.