GENIUSV2: SOFTWARE DESIGN AND MATHEMATICAL FORMULATIONS FOR MULTI-REGION DISCRETE NUCLEAR FUEL CYCLE SIMULATION AND ANALYSIS

by

Kyle Matthew Oliver

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

(Nuclear Engineering and Engineering Physics)

at the

UNIVERSITY OF WISCONSIN–MADISON

2009

Advisor Approval Page

GENIUSV2: SOFTWARE DESIGN AND MATHEMATICAL FORMULATIONS FOR MULTI-REGION DISCRETE NUCLEAR FUEL CYCLE SIMULATION AND ANALYSIS

Kyle Matthew Oliver

APPROVED

Paul P H Wilson

Associate Professor of Engineering Physics

10 June 2009

To my parents, Chris and Joanne, for their quiet but sure love and support (the world has enough cheerleaders, I agree); to my sister, Rachel, who taught me the courage to make hard decisions; and to my friend Carl, who I suspect will get some laughs from the Philadelphia concertgoers with the story of having a nuclear engineering thesis dedicated to him.

ACKNOWLEDGMENTS

I've been at UW-Madison for seven years now, so I've got a lot of people to thank:

- My advisor Paul Wilson, who's been as good a mentor and friend as any advisee could hope for and who's been supportive through all my flailing—good and bad, productive and not-so-much. He also unleashed The Hacker Within.
- My other professors and teachers, especially Greg Moses, Mike Corradini, Doug Henderson, Alex Nagel, Bob Meyer, Ben Liblit, Bob Witt, Rock Mackie, Tom McGlamery, Mike LeMahieu, Susan Hellstrom, and Steve Nathans.
- The other folks who kept me busy in grad school, especially Sandy Courter, Christine Nicometo, Mitchell Nathan, and David Meerman Scott. I couldn't have stayed sane without you and am sorry for those occasions where I gave you the opposite impression.
- My colleagues and friends in and around 414 ERB and CNERG/SANE, especially Arnaud Reveillere, Katy Huff, Royal Elmore, Kerry Haney, Tae Wook Ahn, Matt Terry, Aditya Kumar Pidaparthy, Tracy Radel, Mike Priaulx, and Ryan Grady. Special thanks to Milad Fatenejad, who got me through several crucial roadblocks and taught me how to hack the source, and Ahmad Ibrahim, who absorbed endless grumpiness and has to be the best officemate on campus.
- Chris Juchau and Mary Lou Dunzik-Gougar, who worked on GENIUSv1.
- The Advanced Fuel Cycle Initiative at the U.S. Department of Energy's Office of Nuclear Energy, for funding my work. Special thanks to Mr. Carter "Buzz" Savage, Dr. James Bresee, and Dr. Tom Ward for their guidance and feedback.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Appendix

Page

ABSTRACT

Many factors have recently converged to renew interest in nuclear power in general and advanced nuclear fuel cycles in particular. Because of high technical and socio-economic uncertainty about the nature of future global fuel cycles and the novel technology that will support them, systems analysis and modeling activities have become important aids for fuel cycle planners and policy evaluators. A consensus seems to be emerging that this work cannot precede via methods that decouple the engineering details of the fuel cycle models from the broader international and regional policies that have also historically shaped fuel cycle design decisions. To enable a more integrated approach to fuel cycle systems analysis, the Simulation Institute for Nuclear Energy Modeling and Analysis initiated work on a modeling tool called GENIUS—Global Evaluation of Nuclear Infrastructure Utilization Scenarios. This thesis describes the design of and early methodologies deployed in GENIUS Version 2, a discrete-facilities/discrete-materials nuclear fuel cycle simulation intended to eventually aid in technical analysis and design of advanced fuel cycles as well as in nuclear supply chain robustness evaluation, financial and economic modeling, and non-proliferation and waste management studies. The system model of GENIUSv2 is designed to capture the details of interactions between the various actors in a global hierarchy that includes reactors and other fuel cycle facilities, the institutions that own them, and the regions those institutions serve. A simulation manager facilitates the cooperation necessary for these actors to exchange material in support of reactor operation. After describing the model design and simulation infrastructure of GENIUSv2, this thesis presents optimization-based formulations for two of the formidable problems any discrete-facilities/discrete-materials code must address: (1) how to route materials through the system by matching individual customers for fuel cycle goods and services to appropriate suppliers, and (2) how to properly combine available reprocessed material into recycled fuel with a suitable composition. Finally, it discusses results from a suite of testing and demonstration problems that highlight the code's novel capabilities and summarizes important areas for future development.

Chapter 1

Introduction and context

Controversy over the role of nuclear power, and of advanced nuclear fuel cycles in particular, looms large in discussions of U.S. and global energy policy in light of several diverse but interrelated developments. These factors include the continued increase in global electricity demand[1], emerging scientific consensus regarding anthropogenic contributions to global climate change (Allali et al., 2007), macroeconomic concern about fossil fuel prices and price volatility (Sauter and Awerbuch, 2003), heightened fears about nuclear terrorism due to what political scientist Graham Allison calls "the prism of 9/11" (2005), and the popular notion that a "safe and just solution to the nuclear waste problem" (Darst and Dawson, 2008, p. 19) would improve the chances of success of the so-called nuclear renaissance (see Nuttall, 2005).

This thesis describes the need for, the design of, and key results from a global nuclear fuel cycle systems analysis tool intended to model, evaluate, and eventually optimize various nuclear fuel cycles with respect to, in roughly ascending order of difficulty of the modeling task,

1. their electrical generation capacity,
2. the total mass flows of materials between their various facilities,
3. the isotopic composition of those materials throughout the life of the system,
4. their robustness to perturbations and interruptions of a technical or socio-economic nature, and

[1] Especially in non-OECD (that is, developing) nations, whose projected 2030 generation outstrips OECD nations' by 46 percent (Doman et al., 2008).

5. their total integrated cost.

This kind of simulation and analysis capability is especially important due to the capital-intensive nature of nuclear facilities; uncertainty about the details of their individual operation and their cooperation as a fuel cycle system; and the interdependence of the regional, national, and transnational entities engaged in the nuclear enterprise. However, before describing these challenges and how nuclear fuel cycle systems analysis can be of help in investigating them, we need to be clear about what we mean by advanced fuel cycles and why they're important.

1.1 Advanced nuclear fuel cycles

Nuclear fuel cycles are the systems of facilities that prepare, use, store, and (in some cases) recycle and/or permanently dispose of nuclear fuel and its byproducts. While no generalized description of the nuclear fuel cycle will capture every stage used in every country to support every reactor type, Figure 1.1 sketches a typical proposal for an advanced fuel cycle in which uranium is first mined, enriched, fabricated into fuel, and "burned" in thermal-spectrum light water reactors (LWRs) to produce electricity. The status quo in the United States slates spent LWR fuel for geologic disposal without further irradiation, giving rise to the descriptor *once-through* for fuel cycles like the one we currently use. However, in the figure we can see that the used fuel from those reactors can instead be chemically reprocessed and fabricated into fuel for special fast-spectrum reactors called *burner* or *transmutation* reactors. In theory, this "fast recycle" step can be repeated indefinitely to consume all the fissionable material the system creates, which is why proposed fuel cycles that behave this way are sometimes called *closed* fuel cycles[2].

As an aside, it's worth noting explicitly that there seems to be no universal definition of *advanced* nuclear fuel cycles as such. We can generalize, though, from implicit descriptions presented by a number of closely linked national and international bodies and projects, including the Generation IV International Forum (GIF), the Advanced Fuel Cycle Initiative

[2]For a complete discussion of the integrated nuclear fuel cycle and its component parts, see Cochran and Tsoulfanidis (1990).

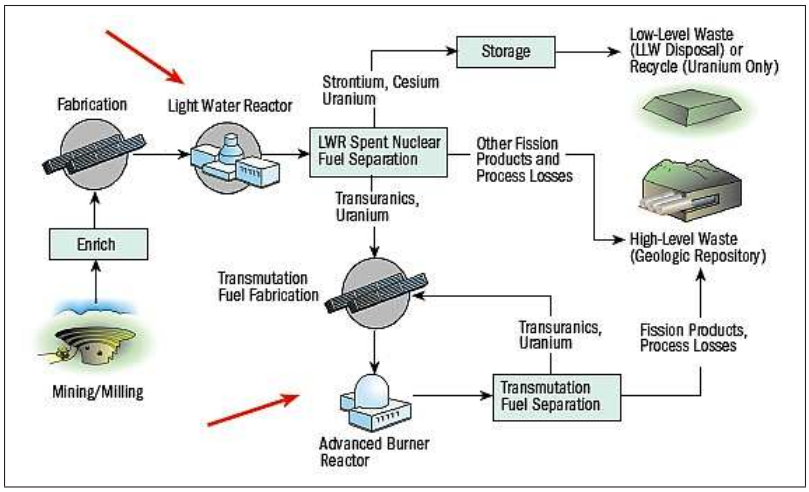

Figure 1.1 Advanced nuclear fuel cycle proposed by the Global Nuclear Energy Partnership program. Repeated actinide recycle in advanced burner reactors improves the performance of the system from a resource extension and waste management perspective, since it recycles material that can produce power and would otherwise end up in a geologic repository. Image from Fischer et al. (2008).

(AFCI), and (most recently) the Global Nuclear Energy Partnership (GNEP). Their materials (GEN IV International Forum, 2009; Kelly and Savage, 2005; Lisowski, 2007) suggest that, if such a definition did exist, it would include the reprocessing of used nuclear fuel for the purposes of waste management and/or resource extension. Thus, in this document the modifier *advanced* is used rather loosely to refer to future fuel cycles that incorporate some kind of recycle.

Some analysts predict that nuclear power will assume a larger role in many nations' electricity generation portfolios in the coming years, partly because of its status as a low-emissions energy source. Indeed, peer-reviewed studies have shown that nuclear generation is among the very best power-producing technologies on an emissions-per-kilowatt basis (Voorspools et al., 2000) and remains competitive as an emissions mitigator when the cost of generation is considered as well (Sims et al., 2003). Predictions of an American nuclear renaissance show early signs of coming to fruition; the Nuclear Regulatory Commission currently expects a total of 22 applications for new plant licenses representing 33 new units (Nuclear Regulatory Commission,

2009). Nuclear power seems poised for a dramatic increase in relevance to national and global energy policy.

However, all this analysis and foreshadowed growth is based on once-through fuel cycle assumptions. The increased attention for *advanced* fuel cycles in the more distant future is due to their potential to greatly extend available uranium resources and to reduce the long-term spent fuel storage burden by recycling the transuranic material created as the fuel is irradiated in reactor cores. Regarding the former benefit, it's interesting to note how resonant the economic and political arguments for continued fast breeder reactor development remain nearly 20 years after the publication of William Jacobi's prescient "Fast Breeder Reactors for Energy Security" (1989); although LWR burnup values have increased in the intervening years, thermal reactors will never utilize more than a couple percent of the uranium that passes through them. This situation represents a clear opportunity for improvement, especially in light of renewed concern over the availability of energy resources. As for waste management and storage, Wigeland and colleagues' important repository benefit study showed the potential to increase drift loading by a factor of dozens to hundreds under the kinds of reprocessing and transmutation schemes that might be possible in a fast burner reactor fuel cycle (2006).

1.2 The role of systems analysis

Needless to say, neither of these fast reactor technologies—nor the requisite reprocessing capabilities necessary to make their fuel—has been commercialized. Indeed, the high cost of nuclear facilities of any kind has long been an anathema to opponents of nuclear technology, a criticism not easily dismissed in an economy that is at present constrained less by carbon than by credit. It's little surprise, then, that funding for advanced fuel cycles is difficult to come by and that an underemphasis on "conservative economics" was one of the reasons cited by a National Research Council committee for their recent unfavorable review of the AFCI and GNEP programs (Board on Energy and Environmental Systems, 2008, p. 71). One way systems analysis activities can contribute to the R&D effort, then, is to provide careful estimates about how much the various proposed fuel cycles will cost.

Perhaps more interesting for the purposes of this thesis, though, were two of the committee's other criticisms, both of which compellingly underscore the importance of nuclear fuel cycle systems modeling and analysis for planning and policy purposes. First, there is great uncertainty about what the mature state of individual GNEP technologies will look like. Fast reactor fuel forms, in particular, were singled out as a major unknown that will likely continue as such for "many years" (Board on Energy and Environmental Systems, 2008, p. 54). Clearly, the successful development of all critical-path technologies is a concern for fuel cycle planners; in the meanwhile, they must depend on flexible system study tools that can adapt to capture the behavior of a variety of potential technologies and the range of dates over which those technologies are likely to become available. In other words, systems analysis tools are useful for examining the consequences of the uncertainty regarding individual fuel cycle technologies.

Following from this first observation is a second and more significant point: uncertainties about the final state of particular technologies have implications beyond the individual facilities they comprise. Designing a fuel cycle *system* requires understanding how the pieces work together. In the face of so much uncertainty, systems analysis becomes important not just for predicting and improving the performance of a fuel cycle but for determining if it will work at all. This is exactly the kind of situation the committee describes when it points out the need to ensure that proposed U.S. recycling methods are compatible with the fuel cycles that other GNEP partner nations are considering (Board on Energy and Environmental Systems, 2008, p. 53).

Finally, to expand our view beyond purely technical uncertainties, we note that most if not all the discussions about advanced fuel cycles involve systems that incorporate significant international cooperation. History and common sense suggest that such cooperation, especially in the field of nuclear materials and technology, is subject to sudden and dramatic change—possibly absent much concern for the technical repercussions. Systems analysis can and should contribute by examining issues of regional interdependence. This kind of uncertainty about the operation of fuel cycle systems is just as important to examine as the more technical concerns. And as we will see in the next chapter, few if any existing tools do that job very well.

We mention finally that we are not the only ones stressing the importance of systems analysis activities in planning the future of nuclear fuel cycles. To close this introductory chapter, we cite at some length the recent National Academy of Sciences/National Research Council report "Internationalization of the Nuclear Fuel Cycle: Goals, Strategies, and Challenges," which also emphasizes the need for an increased emphasis on systems thinking:

> The joint committees believe that a comparison to make choices among different fuel cycle options (reactors, fuel types and sources, spent fuel management, and processing) must use a systems approach. Such analyses would consider the entire life cycle of proposed nuclear energy systems, integrating assessments of fuel processing, fabrication, reactor design, and more. Only in this way can key trade-offs be made among different parts of the system. It is likely that the best technologies for processing spent fuel will be different depending on the specific reactors in which the processed materials will be irradiated, and the fuel fabrication approaches for them . . .
>
> Good decisions among different proposed processing-fabrication-reactor systems require clear, consistent, and well-thought-out criteria, based on justifiable system objectives. Picking a particular numerical target for some system characteristic (such as 99.99 percent purity for uranium separated from spent fuel) without careful analysis of the overall system benefits and costs of meeting that goal leads to poorly optimized systems . . . A good goal would be an integrated reactor fuel cycle system that offers the best combination of economics, safety, security, proliferation resistance, environmental impact, process operability, and sustainability, given the situation that exists for a nation at a particular time.
>
> . . . The role of designers and technical experts is to make clear the choices and trade-offs that need to be made, outline the benefits and downsides of each of the leading approaches, and do their best to ensure that the decisions ultimately made are well informed and carefully considered. (Nuclear and Radiation Studies Board, 2008)

1.3 Thesis overview

Very broadly, the goal of the work I report on in this thesis was to design and implement a nuclear fuel cycle systems analysis tool capable of providing this kind of insight. We can subdivide that overall objective into several subtasks, treatments of which comprise the majority of this document:

1. **Identify key modeling capabilities and software features.** Due to the time-intensive nature of scientific software development and the relative immaturity of fuel cycle systems analysis as a cohesive discipline, careful needs-analysis work should precede any major development effort. Thus, Chapter 2 discusses existing systems analysis tools and their ability to probe questions raised by advanced global fuel cycle proposals. This chapter also introduces the design principles of GENIUSv2, the discrete-facilities/discrete-materials fuel cycle simulation tool we developed to begin to meet these needs.

2. **Design and implement an appropriate fuel cycle model and the infrastructure to support it.** Even with clearly articulated design principles and a list of desired modeling capabilities in hand, the complexity and uncertainty of future fuel cycles ensures that designing an appropriately robust and flexible tool is a non-trivial research and development task. Chapter 3 describes the design and implementation of GENIUSv2 in considerable detail.

3. **Develop mathematical formulations for the two key optimization problems posed by the discrete-facilities/discrete-materials modeling paradigm.** To fully support simulation of both once-through and closed nuclear fuel cycles, a discrete tool like GENIUS must include at least basic functionality for solving two key problems related to system-wide material flow: a routing problem that determines how facilities will work together to mutually satisfy each others' supply of and demand for materials and an approximation problem that determines how collections of separated material can be combined to produce recycled fuel with close to the desired composition. Chapter 4 discusses algorithms for solving these problems using linear and network-flow programming formulations.

4. **Test (and, when possible, benchmark) the code via simple and illustrative cases.** Few fuel cycle modeling problems have unambiguously correct answers, and the emergent system-wide behavior of even seemingly straightforward scenarios can quickly become complex and opaque. Chapter 5 discusses the results of a series of increasingly difficult test problems and compares them to analogous results from other codes, as appropriate.

Chapter 2

Literature review and motivation

This chapter will discuss some of the nuclear fuel cycle systems analysis tools that are currently available, especially their capabilities and some significant gaps therein. It will also introduce the code whose design and methodologies are the focus of this thesis: Global Evaluation of Nuclear Infrastructure Utilization Scenarios, Version 2 (hereafter "GENIUSv2"). Note that this chapter is not meant to definitively survey all of the available tools but, rather,

1. to review relevant portions of the thorough needs analysis work that has been performed elsewhere;

2. to delineate persistent opportunities for GENIUSv2, especially with respect to modeling needs that are unavailable in Idaho National Laboratory's VISION and GENIUSv1 codes; and

3. to introduce the approach and design principles of GENIUSv2.

2.1 Fuel cycle system code surveys

In 2006, Kemal Pasamehmetoglu of Idaho National Laboratory (INL) and Phillip Finck of Argonne National Laboratory (ANL) reported on a collaborative research effort called the Simulation Institute for Nuclear Energy Modeling and Analysis (SINEMA). This project aimed to "develop a simulation network that can model the global nuclear energy infrastructure, the associated fuel cycles and [their] components" (Pasamehmetoglu and Finck, 2006, p. 155). The enterprise-level tool envisioned as both a standalone systems analysis code and an eventual

interface to individual fuel cycle component models was named GENIUS. A prototype of this tool (GENIUSv1) was developed at INL and Idaho State University by Chris Juchau and Mary Lou Dunzik-Gougar (Juchau et al., 2006). The detailed needs analysis and design specification process for GENIUS was originally published by Juchau and Dunzik Gougar as "A Review of Nuclear Fuel Cycle Systems Codes" (Juchau and Dunzik-Gougar, 2006)[1] and was also included in Juchau's Master's thesis about GENIUSv1 (Juchau, 2008). Several points from this review warrant preliminary discussion here.

2.1.1 Facility and material modeling

The proposed SINEMA modeling framework aims to paint a complete picture of the nuclear enterprise and thus to support detailed studies of material transportation, facility operation, fuel cycle system performance, non-proliferation risk, energy economics, etc. Consequently, the GENIUS software specification calls for both facilities and materials to be modeled as discrete entities whose histories can be tracked individually.

Juchau shows that few existing codes have this capability. Most of the others are what he calls *continuous-flow* codes but I will call *fleet-based*, *continuous-flow* codes for reasons that will become clear in Chapter 3 (and will necessitate much of Chapter 5's math). These codes[2] tend to be built on top of commercial "stock-and-flow" systems dynamics packages like Powersim Studio and Stella (Powersim Software, 2008; isee systems, 2008). They operate by modeling the entire fleet of facilities in each stage of the nuclear fuel cycle as a single stock, through which multiple continuously varying streams of material flow in and out at a rate appropriate for the total throughput capacity of the fleet. This modeling decision allows for large fleets of facilities to be modeled within the constraints of the underlying software platform, but it disallows the kinds of discrete interactions required by the SINEMA framework.

[1] A similar but more expansive revisiting of Juchau's basic task is now ongoing at the University of Cincinnati (see Miron, 2008) but was not completed in time to have its results discussed here.

[2] See, for example INL's VISION (Jacobson et al., 2006, 2007) and ANL's DANESS (Van Den Durpel et al., 2003, 2007) codes.

2.1.2 Regional modeling

Implicit in the GENIUS specification's language about its role as a *global* code is the notion that it support modeling of distinct regions around the world. Throughout this document, the term *region* will be used flexibly to refer to intranational, national, and transnational divisions that can be characterized by a distinct time-varying demand for nuclear energy and within which some set of reactors operate in order to satisfy that demand. At minimum, such regional modeling functionality must allow the code to store multiple demand curves (one for each region) and identify distinct subsets of the world's facilities with each region. However, a need exists for regional modeling that is much more rich and complex than would be offered by an accounting-based approach that merely maps demand and facility sets to particular regions and that allows a region's state and behavior to remain largely decoupled from those of other regions.

Chapter 1 introduced the need for systems analysis tools that capture and explore the ways in which regions will cooperate and compete with one another in the context of a global marketplace for fuel cycle materials. Perhaps the most obvious example of this modeling need is the desire to examine various proposals for fuel services arrangements between so-called "supplier states" and "user states," which are illustrated in Figure 2.1. It is unclear at present how and (and perhaps even if) such interactions would work, and beginning to answer those questions seems to us to require a model that captures both the technical details of the reactors and other fuel cycle facilities being operated by both states *and* the economic and financial state of the client region, the user region, and probably other stakeholders in the system as well.

We do not know of any region-enabled codes that do so; Juchau's analysis suggests that Brookhaven National Laboratory's MARKAL code (Loulou et al., 2004) is well equipped to handle the economics but that it models the nuclear fuel cycle in insufficient detail to capture and verify the technical aspects of these interactions. His claim that, with its model of other sectors of the global power industry, MARKAL would be "a good companion" to more detailed nuclear-specific codes resonates with ways in which that code has been used to study the role

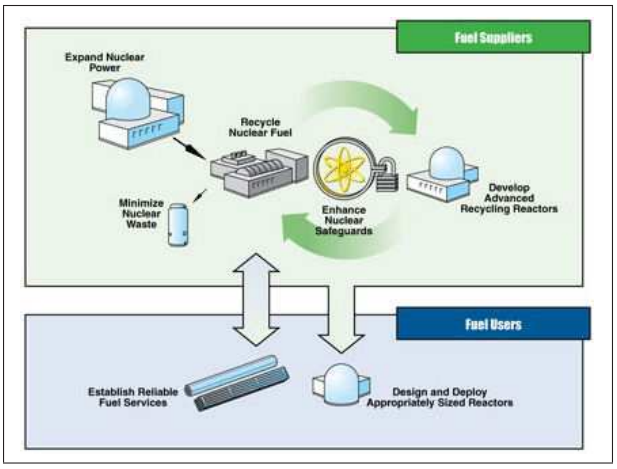

Figure 2.1 Inter-region fuel services interaction proposed by the Global Nuclear Energy Partnership. Investigating economic and diplomatic mechanisms to encourage and secure such interactions is a chief motivator for regional modeling capabilities in nuclear fuel cycle systems analysis. Image from Lisowski (2007).

of nuclear power in the wider context of energy policy and economics[3]. In fact, one hopes this comment will continue to serve as a reminder of where to limit the scope of GENIUS, which may already be an overly ambitious undertaking with respect to the level of detail the model is expected to contain.

2.1.3 Optimization capabilities

From the beginning, GENIUS has been intended for systems analysis *and* optimization. Juchau notes that no available tools optimize the fuel cycle with respect to an integrated, global objective function, though some do perform local optimization on particular parameters. Of course, there are many difficulties associated with formulating nuclear fuel cycle system design (or even just operation) as a robust optimization problem; many of these challenges will be discussed in Chapter 3. Relevant at this juncture, though, is the prevalence in these codes of decision-making heuristics that artificially constrain the design decision space, possibly eliminating the global optimal solution. For instance, a heuristic for choosing an optimal tails fraction at a uranium enrichment plant might not capture the effects of a delay in the enriched

[3] See, for example, Nystrom and Wene (1999).

material's availability to fuel fabricators, resulting in reactor downtime and a much greater increase in the cost of electricity than would have resulted from an enrichment procedure that would have cost a little more but allowed the material to be available sooner.

Jain and Wilson (2006) discuss several of these heuristics (which arise quite naturally in order to handle basic problems like reactor deployment and fuel allocation) and point out that they are present in VISION and DANESS as well as the MIT code CAFCA (Boscher et al., 2004). In fact, it seems impossible to build a functioning code that is completely devoid of decision heuristics. Thus, we merely note at this juncture that their elimination wherever possible (in favor of decision strategies that do respond to a global objective function) is very much an outstanding research question for fuel cycle systems analysts, one to which we will return later in this chapter and especially in Chapter 4.

2.1.4 Software design and infrastructure

The originators of SINEMA understood well the challenges posed by the proliferation of nuclear fuel cycle systems codes and the more detailed codes that model the behavior of the individual facilities and materials that comprise fuel cycle systems. They believed that a unified modeling framework would help manage this phenomenon and coordinate collaboration between developers and, as it were, between the codes themselves . Thus, they specified that, in general, GENIUS should "[h]ave a software architecture that is modular, flexible, open and accessible" and, more specifically, that

1. The tool must maintain abstraction between data and process algorithms. Both databases and modules of process source code should be replaceable or updateable without major alterations to the overall source code and system architecture.

2. The tool architecture, source code and documentation must be as open and accessible as possible to project developers and collaborators, both foreign and domestic.

3. The tool must be able to communicate with other codes through weak-links/databases to support the fuel cycle modeling effort. (Juchau and Dunzik-Gougar, 2006, p. 7)

This is a very difficult specification to meet for codes that weren't designed from the start to meet it, and indeed Juchau and Dunzik Gougar concluded that none of the ones they investigated satisfied the final item (2006, p. 8).

Nevertheless, it is this author's opinion that the promise of a fuel cycle code that meets this group of requirements alone would justify the resources that have thus far been devoted to GENIUS development. One of the major themes of this thesis is that discrete-facilities/discrete-materials (hereafter "DF/DM") fuel cycle systems analysis poses difficult but rich and important problems in the areas of optimization, economic modeling, and data management, in addition to more familiar physics and engineering problems like approximations to in-core isotopic inventory tracking. A tool that can encourage collaboration between the different research groups interested in this problem—especially a tool that supports modular software library substitution for the parts of the fuel cycle that contain tricky but self-contained subproblems—could be a boon to the field. That's what the originators of SINEMA seemed to believe, and the use of modern scientific computing tools may just make it possible. Again, we'll come back to this issue as we introduce GENIUSv2 later in this chapter and in Chapter 3. Let it suffice for now to say that (1) none of the existing tools were designed to support this functionality and (2) we believe figuring out how to provide it represents its own relevant and non-trivial research question.

2.2 Specific limitations of VISION and GENIUSv1

This section discusses two existing codes in extra detail. INL's VISION code is probably the most important existing tool because of its power and maturity, because developers have consistently reported on its progress in the literature (see Jacobson et al., 2006, 2007; Phillips et al., 2007; Yacout et al., 2006b), and because the Department of Energy has made use of it for AFCI- and GNEP-related analysis and projections (see McCarthy, 2007). It also serves as a typical example of fleet-based, continuous-flow systems analysis. The UW-Madison Computational Nuclear Engineering Research Group (CNERG), of which I am a part, has considerable experience using and extending VISION, so we are familiar with its strengths and

limitations and can speak to a few of the areas in which its model is simply incompatible with certain research questions[4].

We do not claim that GENIUSv1 has attained anywhere near VISION's level of sophistication and "market penetration," nor would one expect it to have done so given its age and the resources that have been committed to it. Nevertheless we discuss it here for several reasons. First, it serves as a sort of "proof of principal" for doing DF/DM systems analysis on the kinds of global nuclear fuel cycles proposed by the GNEP program. Second, the difficulties its developers faced have manifested themselves in GENIUSv2 development as well, so examining v1 in a little more detail stands to provide some insight into our own problems. Finally, because GENIUSv1 was designed to fill the needs discussed in Section 2.1, it is important to identify if, where, and why it fell short of that goal, so we can be confident that our work on GENIUSv2 is not redundant.

2.2.1 VISION: A fleet-based, continuous-flow code

We begin this section with a concrete example. Without going into too much detail about the design and implementation of VISION within the Powersim Studio systems dynamics environment, or that of its Stella-based predecessor, DYMOND (Yacout et al., 2006b), we can still achieve some understanding of their basic workings by examining two isolated portions of these models. The top of Figure 2.2 shows the so-called Reactor Park Sector in DYMOND, which is somewhat simpler than its VISION replacement and therefore easier to visualize. Right away, we get some sense for the fleet-based nature of the DYMOND-VISION model; the labeled boxes along the main "conveyor belt" each represent a dynamically calculated value characterizing the number of reactors in each of a number of subsets of the fleet (e.g., reactors that are under construction, reactors that are ready to operate, reactors that are nearing retirement, etc.). The thin arrows show that the rates that determine how quickly reactors move

[4]Though it hopefully goes without saying, it bears explicit mention that this section is not meant as a criticism of VISION. Indeed, many of its features that act as limitations within the framework of our discussion here are strengths in other contexts. And of course it routinely solves many problems that will remain out of reach for GENIUSv2 for some time yet.

from one subset to another (`Final Construction Rate`, `Reactor Aging Rate`, etc.) are controlled by user input parameters (e.g., `Licensing time`) and possibly by feedback from dynamically calculated measures of the system's state. One can imagine how the values in this sector can then subsequently serve to control the behavior of other sections of the model. For instance, we would expect that the number of `Fresh Reactors` will help determine the rate at which material moves in and out of `Fuel in reactors`. This latter value is part of the DYMOND Fuel Cycle Sector, which again is somewhat simpler than its VISION replacement and is therefore shown in the bottom of Figure 2.2 (Yacout et al., 2006a).

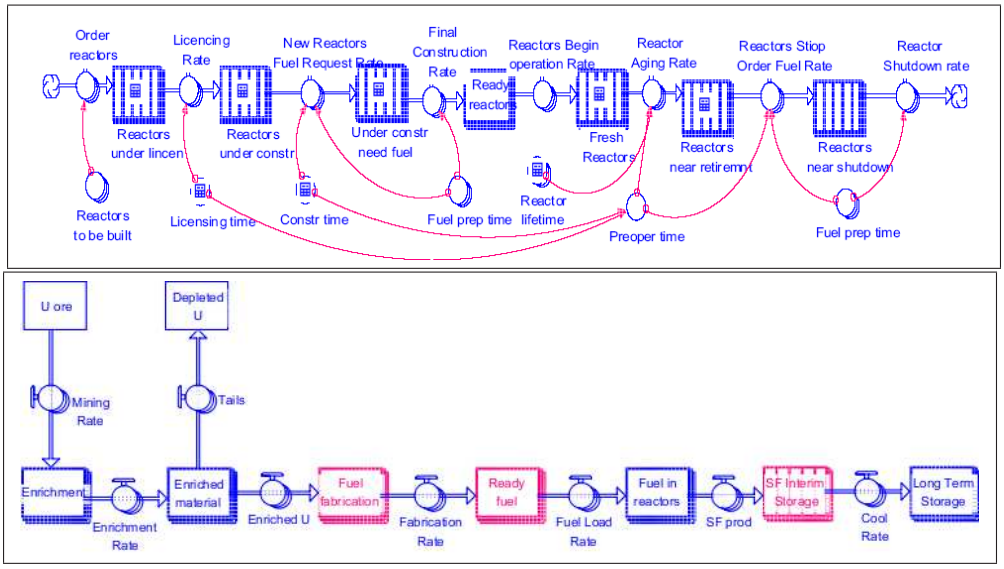

Figure 2.2 DYMOND screen shot showing (top) the code's fleet-based reactor-tracking model (the Reactor Park Sector), including reactor deployment decision heuristic, and (bottom) the code's model of material traveling through the fuel cycle itself (the Fuel Cycle Sector). The VISION equivalents are similar but more complex. Images from Yacout et al. (2006a).

Where does VISION fall short of being able to model advanced fuel cycle problems of the type we're interested in here? Obviously, the biggest gap lies in the lumping of fleets of each type of facility together and tracking only the total aggregate mass flows of each isotope through those lumped entities. Modeling individual facilities in the Powersim Studio framework would quickly become untenable (because of the large number of facilities and linkages

that would need to be manually specified for each study scenario), and modeling discrete material shipments is difficult under the continuous-flow paradigm[5]. Another significant drawback of VISION is the lack of a simple mechanism for performing the kind of regional modeling discussed in Section 2.1.2. This is not to say that such modeling is impossible; one can imagine any number of clever, probably iterative strategies for running individual problems for each region and capturing first-order measures of the way the individual regions would need to interact with one another. However, the model was clearly not designed for capturing the regional heterogeneity of GNEP-type fuel cycles.

Regarding system optimization, note first that the very nature of tools like Powersim and Stella makes especially difficult the would-be optimizer's role of eliminating heuristics. Indeed, these modeling systems are designed to study the effects of feedback and other heuristic decision mechanisms on a dynamic system; in some sense, heuristics go with the territory. At the very least, it seems fair to say that eliminating the general deployment heuristic shown in Figure 2.2 would be a nuisance at this stage in VISION development. Using iterative optimization toolkits to identify promising fuel cycles would requiring doing just that.

Interfacing with those optimization tools, or any other external software for that matter, would also be a nuisance. Indeed, another significant limitation of VISION is the graphical-user-interface-based, stand-alone, closed-source modeling environment. There's very little incentive to develop tools that will work *with* VISION because of the difficulties of linking other software to Powersim Studio[6]. This forces developers to use systems-dynamics-based techniques (or embedded Visual Basic scripts) for solving all of the difficult sub-problems the nuclear fuel cycle presents, even those for which robust and computationally affordable software libraries currently or may someday exist.

We close this section by introducing just such a sub-problem, one of the more difficult ones that fuel cycle analysts face. INL's Steve Piet referred to this problem as the "Winery" issue,

[5] See, for example, the single-reactor benchmark problem discussed in Section 5.1.1.

[6] There are mechanisms for doing so, including the SimCoupler module for linking Powersim models to the Matlab/Simulink modeling framework (Powersim Inc., 2006). But note that this route creates additional dependency on proprietary software.

evoking an illustrative metaphor for the problem (present in all closed fuel cycle simulations) of the "mismatch between the ever-changing composition of used fuel to be separated and the as-fixed-as-possible composition of fuel to be fabricated from recycled [transuranics]" (Piet, 2007, p. 1)[7].

I refer to this problem as *recipe approximation* because I propose approximation-theory-based formulations for solving it (see Section 4.4). Fellow AFCI Fellow Shannon Yi has studied it as well, albeit in the VISION context, and others may be working on it also. The point is that this is a difficult research question that will likely not be settled satisfactorily for some time. In the meanwhile, platforms that make it easy to "plug and play" different forms of the solution (which may require support libraries of their own depending on their level of sophistication) would seem to be ideally suited for testing and comparing promising approaches. VISION's software infrastructure makes it an unlikely candidate for filling this "test bed" role.

2.2.2 GENIUSv1: A discrete-facilities/discrete-materials code

GENIUSv1 filled several of the modeling gaps identified in Juchau and Dunzik Gougar's code review. However, due to limits on time and support, others remained to be addressed. Among its most important successes, GENIUSv1 showed that meaningful mass flow data could be calculated via DF/DM modeling of large, complex fuel cycle scenarios with regional heterogeneity. These scenarios, based on an initial condition representing the current state of the global fuel cycle system and GNEP-type assumptions about future growth and regional interaction, were reported on in Juchau's Master's thesis (Juchau, 2008) and at the Global 2007 fuel cycle conference (Dunzik-Gougar et al., 2007). Figure 2.3 plots some typical output from

[7] The metaphor goes like this:

1. Determine what grapes (used fuel) are available, with what characteristics.
2. Adjust your intended product specifications as needed.
3. If needed, blend different grapes or use a wine cellar to get a more consistent product. Blend grapes (used fuel) from different sources; the properties of those grapes change with aging and source. Use a wine cellar of different wines (separated material) laid down in different years.
4. Determine what you actually get. (Piet, 2007, p. 1)

GENIUSv1: the flow of material to LWR fuel client states during the course of one such simulation.

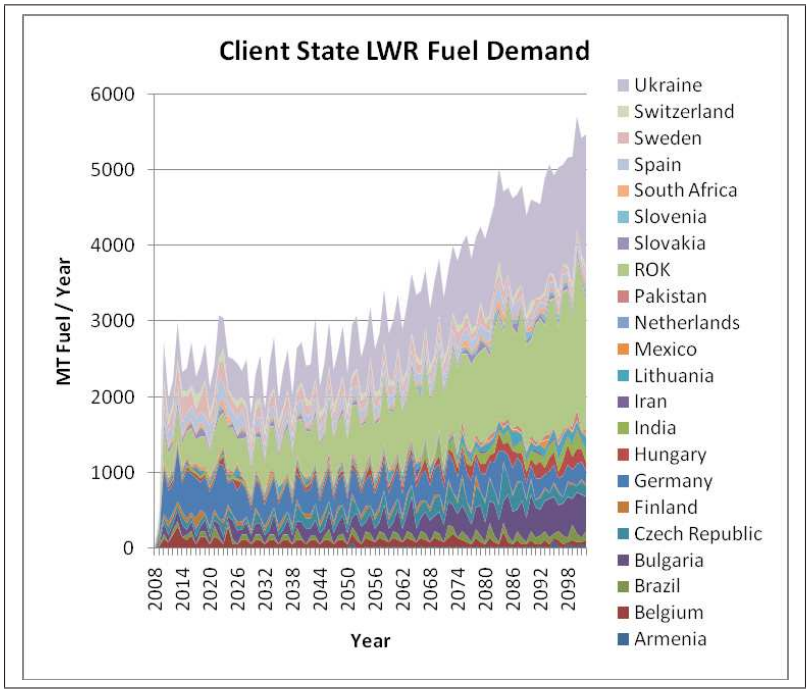

Figure 2.3 Typical demand data from GENIUSv1 plotting the mass flow of LWR fuel to client states (GNEP "user states") under a GNEP-type growth scenario with realistic initial condition. Note that the "jaggedness" of the data is a natural result of DF/DM modeling in which (a) the electricity demand curve does not consider fuel ordering cycles of existing reactors, (b) large, discrete batches of material always travel together, and (c) the visualization procedures do not perform averaging or other artificial smoothing. Image from Juchau (2008).

Note the high level of detail. As material moves from facility to facility, GENIUSv1 sums and prints to the appropriate regional output file the annual mass flow for each stage in the fuel cycle. Thus, collecting data of this nature is a straightforward though perhaps tedious task under the data structures that are in place. However, careful examination of this plot still yields some strategies about ways to move forward. There is, of course, a tension between (on the one hand) the desire to collect as much detailed information as possible about the history of the simulation and (on the other) the ability to efficiently store and process that information. Although I have not performed detailed diagnostics on GENIUSv1, I suspect

that some of its design decisions were made primarily in response to the limitations of this prototype code's software architecture. Relying on the combination of the (relatively) slow and memory-hungry Python scripting language and a system of text-file-based output probably limited the "quantum-size" of GENIUSv1's discrete materials. This limitation manifests itself as one of the causes of the somewhat disconcerting jaggedness of the data in Figure 2.3. As we shall see, GENIUSv2 implements a more robust and detailed system for tracking and storing material histories, one that can handle the decision to track much smaller quanta of material, if desired. Software infrastructure for material tracking is just one of several places where we have gone to great lengths to improve and modernize the GENIUS code.

A second such area is in the potential for realism in the socio-economic interactions between (and, in the case of GENIUSv2, within) the regions of the GENIUS model. To our understanding, the extent of the complexity in modeling these interaction in GENIUSv1 comes in deterministically specifying the fuel cycle region from which each fuel user region will receive its material. GENIUSv2 introduces another layer of abstraction in the hierarchy of fuel cycle actors: the various *institutions* that own the facilities in a given region. It also implements a more flexible and dynamic scheme for modeling the interactions between various facilities, institutions, and regions (see Section 3.1).

That scheme is also more optimization-friendly, which leads us to the third novel improvement in GENIUSv2 with respect to its predecessor: the elimination of two major decision heuristics. The first is the GENIUSv1 solution to what we will refer to as the *fuel cycle design problem* (FDP). GENIUSv1 still deploys fuel cycle facilities in a manner similar to VISION, by hard-coding some reasonable set of feedback-based rules describing the conditions under which a new facility should be built. For instance, fast reactors are deployed in response to a set of constraints that include regional electricity demand and the availability of recycled material for fuel. GENIUSv2 eliminates the FDP heuristic by changing from a demand-driven facility deployment to a user-driven one.

The second GENIUSv1 decision heuristic we've eliminated arises only under the DF/DM approach. Unlike in the in the fleet-based, continuous-flow paradigm, in which a material flows

through each stage of a single, unified pipeline (like DYMOND's Fuel Cycle Sector from Figure 2.2.), the control logic in a DF/DM code must constantly ask the question "Which supplier of commodity *X* should customer *Y* get its fuel from?" We'll call this the *materials routing problem* (MRP). Juchau solves it via a combination of the user-specified, region-to-region pairings discussed earlier or, when no pairing exists, a simple heuristic that matches each customer to the supplier that currently has the most unused capacity. This heuristic is perfectly reasonable, but it precludes an optimization-based approach that can capture the consequences of market competition and other socio-economic forces. We replace it with an optimization problem known as a *network flow program* that solves the MRP with regard to a global objective function: the total cost of the chosen routing (see Section 4.2).

2.3 GENIUSv2 design principles

We can generalize this chapter's findings into a set of four design principles that have guided my work on GENIUSv2. These principles are drawn from the needs analysis work summarized in this chapter, but they also reflect my understanding of the most important and challenging among those needs, especially the ones GENIUSv1 does not meet. These principles are general enough to loosely summarize the overarching themes of the very detailed GENIUS specification (see Juchau and Dunzik-Gougar, 2006) but have proved to be a bit more supple than that document in guiding day-to-day design and implementation decisions. I recently mentioned these four principles in a conference paper currently in press, calling for system study tools that are simultaneously *detailed*, *flexible*, *robust*, and *generative*:

> By *detailed*, we mean that [the study tools] model a wide range of information about the nuclear fuel cycle scenarios being considered. Detailed study tools must model very specific facility deployments and facility operation modes. By *flexible*, we mean that they adapt well to new approaches for how those facilities should work together. Flexible tools are as free as possible from assumptions about what an advanced fuel cycle flowsheet will look like and are easy to modify or augment to model new approaches. By *robust*, we mean that they store and process the necessarily large data sets efficiently. Robust tools should use modern computing libraries and other resources. By *generative*, we mean that they can identify new,

> promising, or optimal fuel cycle scenarios. A generative system analysis tool is in a sense also a system design tool. (Oliver et al., 2009)

2.4 Summary

This chapter introduced the GENIUS project and surveyed the work that helped determine its purpose and scope. We noted gaps in the modeling and optimization capabilities of existing fuel cycle codes, especially with respect to the almost certainly international nature of future advanced fuel cycles (including those proposed by the GNEP program) and the difficult problem of limiting dependence on decision heuristics. Particular attention was paid to INL's popular VISION code, which is not compatible with the discrete-facilities/discrete-materials modeling paradigm called for by the SINEMA program, and to GENIUSv1, which served as a useful prototype but needs substantial revision in order to be compatible with the overarching goals of the project. These claims and observations were then generalized into a set of design principles for GENIUSv2. The next chapter will show how those principles manifested themselves into an actual code.

Chapter 3

Modeling and design

This chapter describes the design of GENIUSv2, with particular emphasis on its model of the nuclear fuel cycle. Neither this chapter nor the thesis as a whole is intended as code documentation or as a user guide or other systematic summary of GENIUSv2. I will focus on how the code provides (or is designed to be extended to provide) many of the novel modeling and analysis capabilities identified in Chapter 2. Although optimization considerations affected many of the design decisions explained in this chapter, formulations for the various optimization problems this model gives rise to are found in Chapter 4.

Section 3.1 will introduce the nuclear fuel cycle system model itself. This discussion will include descriptions of the classes that define the behavior of its discrete facilities and discrete materials, in addition to the other entities the model accounts for. Section 3.2 will describe the simulation machinery, that is, the aspects of the code that allow the entities in the model to work together to produce the kinds of system behavior we want to study. This machinery includes a *timer* that moves the simulation forward, a *bookkeeper* that helps record the history of the simulation, and a *manager* that oversees and directs cooperation between the entities in the model. Finally, Section 3.3 describes the infrastructure of the code, including the use of modern computing tools for managing the large amount of data a DF/DM code requires and produces.

3.1 System model

The global nuclear fuel cycle model implemented in GENIUSv2 was first described at the 2007 GNEP Annual Meeting (Wilson and Oliver, 2007) and has changed very little in the intervening time. The model's hierarchical structure is illustrated in Figure 3.1 via a simple two-region example. Each scenario modeled will include at least one instance of three types of discrete entities: *regions* subject to some demand for nuclear power, *facilities* that attempt to provide that power or to otherwise support the fuel cycle globally or regionally, and the *institutions* that own those facilities. So in Figure 3.1, the region on the right might represent a large fuel cycle state in which many different institutions sell electricity or nuclear fuel cycle materials or services. The left-hand region might represent a client state with a single government-owned utility that operates reactors fueled with material purchased or leased from the fuel cycle state.

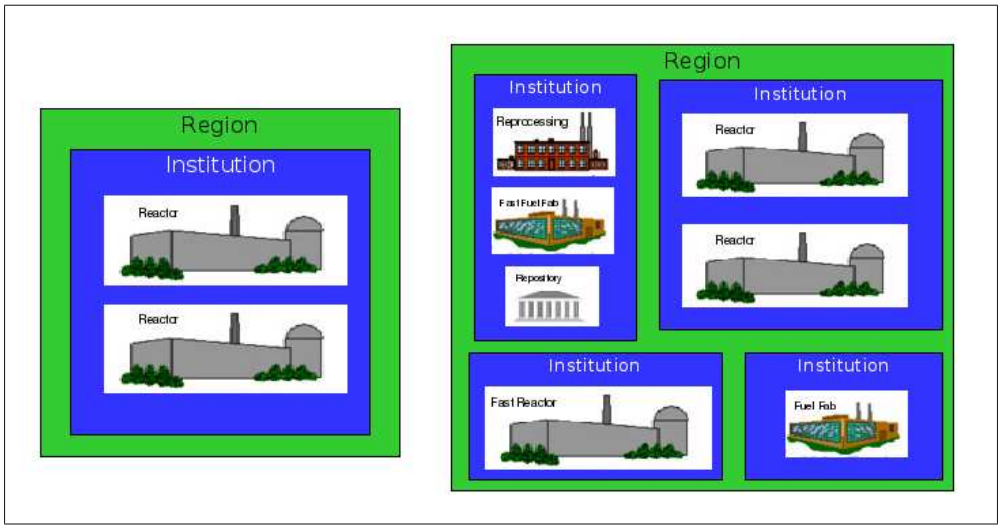

Figure 3.1 The hierarchical structure of the GENIUSv2 model. Note the addition of a new "layer" of discrete entities: the institutions that own and operate nuclear fuel cycle facilities. Image adapted from Wilson and Oliver (2007).

The terminology for naming these entities has been chosen with care and reflects our desire that the model be as flexible as possible. In typical use, such as for the kinds of problems Juchau ran with GENIUSv1, a region will represent a particular country. However, one can imagine

cases in which a number of countries in a transnational region might have energy infrastructures that are sufficiently coupled to warrant modeling them as a single discrete entity[1].

Similarly, we use *institution* as a general term to encompass utilities and other public, private, or governmental companies or organizations that participate in the nuclear fuel cycle. Note that many (if not most) of these institutions own more than one facility. As I mentioned in Section 2.2.2, institutional modeling is new to the GENIUS project and—at least to our knowledge—to the entire collection of codes that perform detailed nuclear fuel cycle modeling and systems analysis. We believe this intermediate layer will be necessary (or at least highly useful and appealingly realistic) for the kind of coupled technical and economic analysis that will be needed to investigate global fuel cycle proposals in the manner suggested by the National Academies (see Nuclear and Radiation Studies Board, 2008).

Finally, note that the term *facilities* emphasizes that, although reactors, fuel fabrication plants, enrichment plants, etc. all have specialized data and behaviors, nuclear fuel cycle facilities look very much alike, conceptually. Except for those at the extreme ends of the fuel cycle (uranium mines and waste repositories), each must obtain a particular fuel cycle material (e.g., fresh fuel or unenriched uranium hexafluoride), perform some operation on it (e.g., irradiation or enrichment), and pass along new materials (e.g., spent fuel or enriched uranium hexafluoride) to another interested facility.

This functional similarity, and others like it, motivated our extensive use of *inheritance*[2] throughout the region-institution-facility (hereafter "R-I-F") hierarchy, as well as in the material model discussed in Section 3.1.3 below. In GENIUSv2, the discrete facilities that so far we've only discussed theoretically are in fact implemented as instances of several specialized subclasses of a more general facility class. Similarly, the discrete materials are instances of the material class and its subclasses. These two class hierarchies take literal advantage of inheritance to share common data and behaviors. However, the general idea of inheritance operates in the three levels of the GENIUSv2 R-I-F hierarchy as well. For instance, the economically

[1] Or, conversely, an intra-national region might operate independently enough to justify distinct treatment.

[2] See Appendix A for a brief description of this and other important terms from object-oriented programming.

viable operation of a facility depends upon a number of financial parameters, including material costs, tax rates, debt costs, etc. The values of some of these parameters (insurance costs, say) might be a function of the facility itself, but others (such as the internal rate of return) are a function of who owns the facility, and others still (such as the tax rate) depend mostly on where the facility is located. Individual facilities can then "inherit" (in a non-technical sense) the appropriate values based on their membership in a particular region and institution.

3.1.1 The region and institution classes

At this point in the development of GENIUSv2, the main jobs of the region and institution classes are storage and message-passing. The latter will be discussed in Section 3.2.3 for all of the classes that have the ability to communicate. As for storage, the region class must provide each region object with data structures for storing the electricity demand of the region and the collection of institutions that operate within its borders, so to speak. Because we decouple electricity demand from facility deployment for optimization purposes and do not currently include the regional electricity demand in the objective function for determining materials routing, the storage of regional demand data is not, strictly speaking, necessary at present. However, as the code's objective function formulations continue to mature, it will be important to have access to regional electricity demand at run time. The institution class has similar storage needs, with each institution keeping track of the facilities it operates and its static plan for building new ones in the future. As I mentioned above, the region and institution classes will also store relevant economic and financial parameters for their members to inherit as the GENIUSv2 model becomes more sophisticated.

Finally, though regions and institutions need not be implemented as classes in order to support this particular functionality, we note also that the R-I-F hierarchy naturally establishes set-theoretical relationships between simulation entities; for instance, a facility is a member of (1) its own set, (2) the set of facilities owned by the institution it belongs to, and (3) the set of facilities residing in a given region. Thus, the region and institution classes (and also the facility class described below) implement set-theoretical operators (`isSubsetOf()`,

`isSupersetOf()`, etc.) to allow for simple and efficient calculation of these relationships when exploring the possibility of trade between the various simulation entities.

3.1.2 The facility class

The facility class and its various subclasses are more complicated than regions and institutions because there is much more that a facility needs to know and be able to do, so to speak. However, as I mentioned above, it's possible to simplify and appropriately encapsulate some of this complexity by including common data and behaviors in an abstract facility class and specialized behaviors in the appropriate subclasses. We first discuss the common facility class. The following prose descriptions are summarized and augmented in a tabular listing (Table 3.1) at the end of this section[3].

3.1.2.1 Facility data

The simplest aspect of the facility class to understand is the set of data that describes a facility's history and operation. For instance, all facilities have some characteristic construction time, some time at which they begin operating, some periodic cycle time that describes the length of their basic unit operation, and some time at which they're decommissioned. These and other data that apply to all facilities (though their values may vary from subclass to subclass and even from object to object instantiated from those subclasses) comprise the member data of the facility class. Other important types of facility data include capacities, capacity factors, and financial parameters.

[3]This table and others like it are not meant to provide an exhaustive listing of every member in the facility class definition, and neither will the entire source code for GENIUSv2 be given in an appendix. (At present time, the GENIUSv2 source code comprises more than 17,000 lines, not counting the Python pre- and post-processors.) However, it is our intention that GENIUSv2 be available as open-source software, though we have not yet settled on a distribution method. See the Computational Nuclear Engineering Research Group home page, `http://cnerg.engr.wisc.edu`, for current information.

3.1.2.2 Material storage

We need not understand much about the GENIUSv2 material classes to understand that facilities will need assorted data structures for keeping track of all the discrete material objects they will pass between one another. These data structures can be divided into two types that we'll call *storage buffers* and *process lines*. The latter are the easier to understand; they represent the materials currently being operated on by the facility. For instance, materials stored in the operating line at an enrichment plant can be thought of as being "in the cascade" of whatever sequential enrichment technology is employed at that particular facility. If we recall the observation that a facility is a black box into which raw materials flow and out of which refined products emerge, the process lines are the data structures that hold the relevant materials during their time in the box.

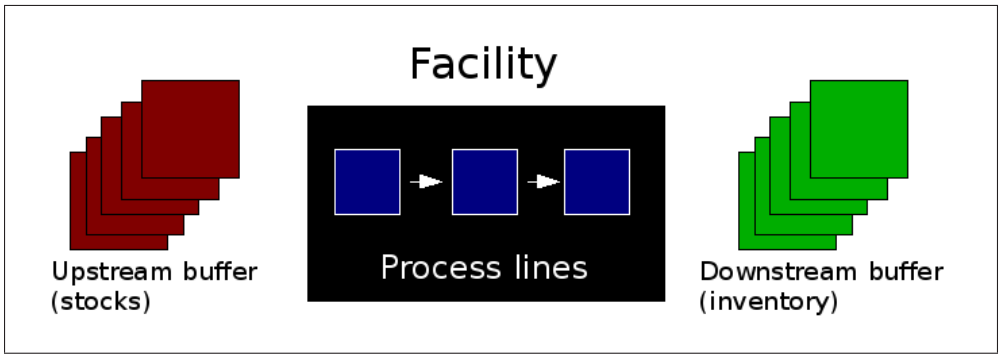

Figure 3.2 An illustration of the two types of material-storage data structures in the GENIUSv2 facility classes. An upstream buffer called the stocks can store materials that have arrived at a facility but are not yet ready for processing. Once a material has been processed (during which time it is stored in the process lines), it can be sent to a downstream buffer called the inventory.

On the other hand, a storage buffer can be thought of as a collection of either raw materials or processed materials that are *waiting*, respectively, to be processed at this facility or to be sent along to the next one. In other words, buffers are the collections of material piled up on either end of the black box (see Figure 3.2). Under a "just in time" model, these buffers would not be necessary; however, they allow us to simulate the risk aversion (and sometimes decay heat reduction) strategies practiced by real-world nuclear fuel cycle facilities—facilities that, after

all, comprise an expensive and sometimes very time-sensitive supply chain. Including buffers in the facility design will allow GENIUSv2 developers and users to investigate strategies for ensuring system robustness through wise maintenance of and reliance on material buffers. We hope that, once these procedures are better understood, they can be parameterized and included in policy optimization studies.

3.1.2.3 Operational methods

Just as all facilities share common data, so too do they share common methods[4] for going about their business as facilities. These methods include procedures for beginning a cycle, submitting offers and requests for material, sending material to other facilities, receiving material from other facilities, etc. Developers can decide whether a facility subclass should use the general facility method for performing an action (for instance, the general methods for sending and receiving material are likely sufficient for most subclasses) or to override the generic facility behavior (methods for *ordering* material likely depend on the specialized role of the facility subclass, since that role determines the type and amount of material to be ordered).

[4] See Appendix A.

Table 3.1 Selections from the facility class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	Description
`int ID`	The unique identifier for this facility.
`FacType myType`	The type enumeration for this facility (i.e., MM for mine/mill, FF for fuel fab, etc.).
`int constrTime`	The time it takes to construct this facility, in months.
`int lifeTime`	The lifetime for which this facility can operate, in months.
`int processTime`	The time, in months, that it takes for this facility to complete a process cycle.
`double capacity`	The monthly capacity of this facility (units vary).
`double capFactor`	The capacity factor for this facility during this time step.
`int startContr`	The simulation time at which this facility started being built.
`int startOp`	The simulation time at which this facility started operating.
`int startCurrProc`	The simulation time at which this facility started this cycle.
`deque<Material*> stocks`	The buffer of materials present at this facility and waiting to be operated upon.
`deque<Material*> inventory`	The buffer of materials present at this facility and waiting to be sent elsewhere.
`ProcessLine ordersExecuting`	A complex data structure that stores the materials this facility is operating on, along with information about the process.

Member functions

Declaration	Description
`void beginCycle()`	Begins this facility's next operational cycle, offering and requesting new material, as appropriate.
`void sendMaterial()`	Sends material to a given facility, according to some set of instructions.
`void receiveMaterial()`	Receives the given material, placing it on the stocks or using it immediately, as appropriate.
`void decommission()`	Decommissions this facility.

3.1.3 The material classes

Before we discuss the facility subclasses, which perform specialized operations on materials, we need to introduce the classes from which those material objects are instantiated. Per the various GENIUS requirement review documents, we need material objects to store enough information so that after execution we can reconstruct two important data sets: the isotopic history and the facility history of every material object in the simulation. Because there can easily be hundreds of thousands or even millions of material objects instantiated during large simulations[5], it is important that they store this information efficiently. One obvious answer to this challenge is to only record composition or location data when this information *changes*. We can further improve the data storage outlook by storing material compositions sparsely—as a map of isotope codes to the corresponding mass or number of atoms, instead of as a vector where each vector index corresponds to one isotope in a predetermined list to be tracked. The vector-based approach would be slightly faster and easier to work with, but it detracts from the code's flexibility by committing to a hard-coded isotope list, and it unnecessarily stores large numbers of zero entries (many simulation objects contain only a handful of isotopes).

An interesting tension emerges from the above design scheme in the case of materials containing non-stable isotopes. From a macroscopic point of view, such materials' compositions change continuously, which would seem to necessitate month-by-month radioactive decay calculations. While such an approach is expensive but not prohibitive in continuous-flow codes, it's both in the case where we must perform the calculations not on dozens of stocks and flows but on the hundreds of thousands of discrete objects that have emerged from reactor cores and thus contain dozens of radioactive constituents. To resolve this tension, we observe that we actually care about the composition of a given radioactive material at only a few points along the back end of the fuel cycle[6]. Thus, we adopt a *decay-on-demand* strategy. The material class implements a method that first calculates the number of months since it last recorded its

[5]For example, in a simulation where 1,000 pressurized water reactors operate simultaneously, there will be almost 200,000 GENIUS material objects stored in memory to represent just the fuel assemblies currently residing in those reactors.

[6]For instance, before shipping an object representing used fuel from on-site storage to a reprocessing plant or off-site storage facility, a reactor may wish to calculate that fuel's instantaneous decay heat. Similarly, a

composition and then performs the calculation to simulate decay over that number of months. Although this approach still results in "dense" composition histories for materials residing in "decay-sensitive" parts of the fuel cycle, it nevertheless offers significant performance gains and allows the model to capture the important effects of decay[7]. See the GENIUS source code and documentation for specifics about the actual numerical method used to perform decay calculations, which was implemented and tested by UW-Madison Computational Nuclear Engineering Research Group (CNERG) member Kerry Dunn.

The other important piece of information stored by material objects is their *commodity* type. We appeal here not to the rigorous economics definition of commodity but to something closer to the operations research usage, which refers to network programs (see 4.2) as being either *single-* or *multi*-commodity problems depending on whether flows on the network are of a single homogeneous material or multiple material types. The commodity enumeration within GENIUSv2 is used throughout the code; most importantly, it's the mechanism by which facilities specify the kinds of materials they wish to request from or offer to one another. The features of the material class discussed so far are summarized in Table 3.2.

The material class hierarchy currently includes a single specialized subclass whose purpose is to represent fuel *assemblies*. Assemblies are stored and passed between facilities in special custom containers called *batches*, which of course correspond to the fuel batches that reactors order to replace a subset of the assemblies within the core during each refueling cycle (see Cochran and Tsoulfanidis, 1990). We track individual fuel assemblies within GENIUSv2 in accordance with our desire to model the fuel cycle in as much detail as possible, but we acknowledge that writing out the histories of each individual assembly (as opposed to the more typical practice of writing out batch-wise histories) increases the size of the output dataset

fabricator of recycled fuel will wish to know the exact composition of the material from which it constructs new fuel assemblies.

[7]Juchau reported in his thesis that neither GENIUSv1 nor CAFCA perform explicit radioactive decay calculations. Alone among discrete-materials codes supporting this capability at the time were the French Atomic Energy Commission's COSI and Eric Schneider's Comprehensive Physical and Economic Model of the Nuclear Fuel Cycle (Juchau, 2008).

Table 3.2 Selections from the material class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	Description
`long ID`	The unique identifier for this material.
`Commodity myType`	An enumeration representing the commodity type of this material (yellowcake, enriched uranium hexafluoride, etc.).
`CompHistory compHist`	A complex data structure representing the complete isotopic history of this material. Implemented as a map that pairs the times at which the composition changed with the new composition at those times.
`FacHistory facHist`	A complex data structure representing the complete facility history of this material. Implemented as a map that pairs the times at which the location changed with a record of the source and destination facility identifiers.

Member functions

Declaration	Description
`void changeComp()`	Changes this material's current composition as specified in the function arguments.
`void logTrans()`	Logs a transfer of this material between two facilities.
`void absorb()`	Absorbs the given material object into this one. Effectively an addition operator for materials.
`void extract()`	Extracts the contents of the given material object from this one. Effectively a subtraction operator for materials.
`void decay()`	Decays this material's composition for the number of months since its last composition change.

by two to three orders of magnitude, with performance slowdowns to match. Collecting assemblies into batches, which store aggregate data representing the sum of their constituent assemblies' composition histories and a single copy of their collective location history, allows GENIUS to support the user's choice of either batch-wise or assembly-wise output reporting. As one might therefore expect, the assembly class (see Table 3.3) includes only one additional assembly-specific member, and the batch class (see Table 3.3) looks very much like the material class, with the exception of a few specialized functions and data structures for storing and managing the assemblies themselves.

Table 3.3 Selections from the assembly class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	Description
[Inherited data]	See superclass (material) data, Table 3.2.
`stack<pair<int, int> > batchTracking`	A stack that stores pairs comprising a batch number and the time when this assembly joined it.

Member functions

Declaration	Description
[Inherited functions]	See superclass (material) functions, Table 3.2.

Table 3.4 Selections from the batch class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	Description
`long ID`	The unique identifier for this batch.
`CompHistory compHist`	Like the material class's compHist, but summed over all the assemblies in this batch.
`FacHistory facHist`	Like material class's facHist, but for the batch as a whole.
`int numAssems`	The number of assemblies in this batch.
`FuelArray myFuel`	A complex data structure that stores the assemblies in this batch as they might be arranged in a reactor core.

Member functions

Declaration	Description
`void changeComp()`	Changes this batch's current composition as specified in the function arguments.
`void logTrans()`	Logs a transfer of this batch between two facilities.
`void transmute()`	Changes the composition of this batch and its assemblies in order to simulate irradiation in a reactor core.

3.1.4 The facility subclasses

The specialized behavior of most facility subclasses is fairly intuitive given an overall understanding of fuel cycle operation (see Figure 1.1). For instance, most facilities have a specialized method for performing their particular processing step on a material object of the appropriate type (mines have a `mine()` method, conversion facilities have a `convert()` method, and so on). In those cases where a specialized datum or set of data is necessary to support

subclass-specific operation, these are also data members of the subclass. For instance, enrichment plants include a data member representing the U-235 enrichment of their waste stream[8], which at present is assigned by the user but could also be set dynamically via feedback or an optimization routine, as appropriate. More complete information about the GENIUSv2 facility subclasses is available in the online code documentation (Oliver et al., 2008), but the following sections discuss the three most challenging and atypical subclasses: the reactor, separations, and repository classes.

Note briefly that the bulk of the methodological work performed for this thesis is directed toward getting these facilities to work together, not toward maximizing the performance or realism of how any one particular type is modeled. Many of the assumptions and simplifications that are generally deemed necessarily in fuel cycle systems codes are present here as well. This section aims only to explain the important details *currently* implemented, in order to provide the reader with enough background to place the results of Chapter 5 in their proper perspective. The means by which inter-facility cooperation is implemented is meant to be reasonably robust to changes in the internal behavior of each facility type. Thus, the shortcomings documented in this section can be addressed as necessary without, we believe, major code-wide disruption in the future.

3.1.4.1 The reactor class

The reactor class is more complex than most facilities due to the nature of (1) the materials that reactors operate on and (2) the burnup operation itself. Whereas most fuel cycle facilities change the composition of single material objects according to explicit closed-form results, reactors operate on batches of materials that emerge from reactor cores with transmuted isotopic compositions that are difficult to predict. Handling Problem 1 is simple enough; Table 3.5 shows that the reactor class includes a specialized process line (`currCore`) and buffers (`batchStocks` and `batchInventory`) that store batches of material rather than individual quanta. A thorough treatment of Problem 2, on the other hand, would require a thesis in

[8]The so-called *tails fraction* (see Benedict and Pigford, 1981).

and of itself[9]. Thus, the discussion here is limited to defining the problem, describing how GENIUSv2 implements the standard highly simplified solution, and referencing some promising new work that could be incorporated into GENIUSv2 in the future.

Table 3.5 Selections from the reactor class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	**Description**
[Inherited data]	See superclass (facility) data, Table 3.1.
`queue<Batch*> batchStocks`	A buffer of batches present at this reactor and waiting to be put into the reactor core.
`deque<Batch*> batchInventory`	A buffer of batches present at this reactor and waiting to be sent to separations or long-term storage.
`queue<Batch*> currCore`	The "process lines" for a reactor, namely, the core itself.

Member functions

Declaration	**Description**
[Inherited functions]	See superclass (facility) functions, Table 3.1.
`void decommission()`	Decommissions this reactor. Overrides the facility version of this function in order to empty reactors' extra buffers.

From our modular perspective of fuel cycle facilities as black boxes that operate on material objects (see Figure 3.2), our task is always to define a procedure, T, subject to reasonable constraints imposed by the relevant chemistry or physics, to transform a set of M feed material composition vectors, $\{C_{in}^m\}$, into N output composition vectors, $\{C_{out}^n\}$, via some amount of work, Z:

$$T_Z(\{C_{in}^1, \ldots, C_{in}^M\}) \Rightarrow \{C_{out}^1, \ldots, C_{out}^N\} \tag{3.1}$$

Our hope is always that we can model this transformation via closed-form equations of some small number of variables and parameters and that Z can help us measure the time the process takes.

Let's illustrate this procedure with a simple concrete example. In the case of enrichment plants, we are fortunate that a simple mass-balance derivation provides just the kind of equation

[9]Indeed, half of Yi's master's thesis is devoted to formulations for transmutation modeling suitable for fuel cycle systems analysis codes (2008).

we need, at least in the case where we can neglect all isotopes other than U-235 and U-238[10]. If an enrichment plant is asked to produce a material object containing P tons of uranium enriched to a U-235 mass fraction of x_p from a suitable feed material object of mass F and U-235 mass fraction x_F, and if the user or an optimization routine assigns our plant to operate with a tails fraction (waste stream enrichment) x_w, then the C_{in} and C_{out} vectors are completely determined by P, F, the x's, and the waste object mass, $W = F - P$. Plus, the required facility throughput can be calculated, in ton-SWU, as follows:

$$\begin{aligned} Z = P[&(2x_p - 1)log(\frac{x_p}{1 - x_p}) \\ &+ (2x_w - 1)log(\frac{x_w}{1 - x_w})\frac{x_p - x_f}{x_f - x_w} \\ &+ (2x_f - 1)log(\frac{x_f}{1 - x_f})\frac{x_p - x_f}{x_f - x_w}] \end{aligned} \tag{3.2}$$

If this job were the only one being processed at a given time, we could then say that it would require $\frac{Z}{Z_{month}}$ months to complete, where z_{month} is the monthly capacity of the plant.

Unfortunately, far from being completely determined by just a few terms in a set of simple, closed-form equations, the output isotopic vectors for the materials that emerge from irradiation in a nuclear reactor vary widely and require expensive computation via dedicated codes many times more advanced than GENIUS. They depend differentially on the power level of the reactor, the neutron's energy spectrum during operation, and many core-specific design parameters, some of which change significantly over the life of the core. Because systems analysis codes are expected to model hundreds if not thousands of reactors over the course of the dozens of core refueling cycles that occur during each one's lifetime, direct coupling to core physics simulation codes is not computationally feasible—nor do we really desire this level of detail in a model of the system as a whole.

[10] A sensible assumption when dealing with non-recycled uranium. For recycled uranium, de la Garza's *matched R cascade* model (1977) and Benedict and Pigford's subsequent extension (1981) are suitable. Unfortunately, they are a bit harder to implement, since they lack a closed-form solution and must be computed numerically.

The standard short-term solution in other codes has been termed the *recipe*-based approach. Developers choose several representative reactor types and perform full core-physics calculations in each type for a number of typical fresh fuel composition vectors, termed *input recipes*, representing the core as a whole. Each input recipe is paired with a corresponding *output recipe* extracted from the solution of the complete fuel irradiation and transmutation problem. Thus, reactors simulated in the code must choose from among the available input recipes and must assume reactor operation in accordance with the assumptions that determined the corresponding output recipes.

The matter of somehow calculating the reactor's work (i.e., the power it produces) presents an additional complication. The appropriate measure is fuel burnup, the quantity of energy produced per fuel mass, expressed in the units gigawatt-days per metric ton of heavy metal $\frac{GWd}{tHM}$. However, this quantity also depends complexly on the minute details of reactor operation and cannot be expressed as a simple linear function of core residence time. Thus, we include multiple input-output recipe pairs, each representing a given fresh fuel composition irradiated to a given burnup. This scheme is illustrated in Figure 3.3 using the notation from Equation 3.1. Such a scheme removes several important degrees of freedom and requires careful accounting to produce strictly correct answers for power production. For now, GENIUSv2 relies on a small set of hard-coded recipes chosen for benchmarking purposes (see Chapter 5) and in one case provided by a client. Let it suffice to say that, though these data members were omitted from Table 3.4 for clarity, the batch class stores a recipe identifier so that its `transmute()` method can produce the correct output isotopics when batches are removed from reactor cores. The user assigns an appropriate choice of input and output recipe for the fresh fuel each reactor will order and the used fuel it will produce. This assignment implies a *de facto* burnup, which, depending on the user's care, may or may not be consistent with the user-assigned reactor cycle times and power capacity. Eliminating this potential inconsistency should be a top priority as GENIUS development moves forward.

Promising research has been proceeding elsewhere to more satisfyingly solve the reactor transmutation problem. Yi recently (2008) extended work by Hermann and Westfall (1998) to

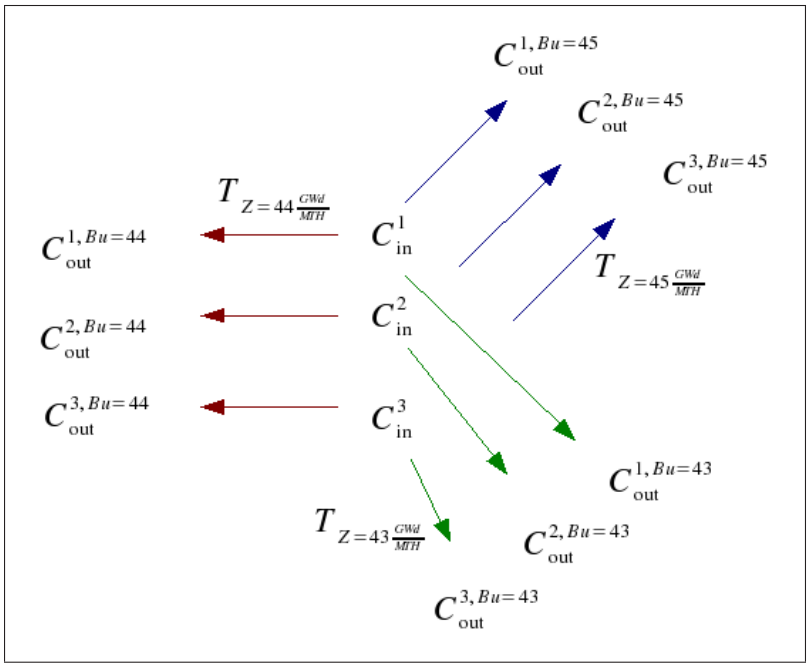

Figure 3.3 Schematic of the standard recipe-based transmutation that materials undergo in reactors, as currently implemented in GENIUSv2. Each input composition is mapped to an output composition by assuming some characteristic burnup when assigning an in-out recipe pair to a reactor.

develop an algorithm suitable for fuel cycle systems analysis codes, and Scopatz and Schneider (2009) have an article in press describing a different method. One or both of these approaches may prove feasible for future adaptation for or direct application in GENIUSv2.

3.1.4.2 The separations class

While not as complex as reactor transmutation, the chemical reprocessing operations present in fuel cycles that employ used fuel recycling pose another non-trivial modeling task. We handle it in the GENIUSv2 separations class, instantiations of which take in used fuel from reactors and produce material objects that represent either useful feed materials for fabricators of recycled fuel or waste materials destined for storage or disposal.

There is, of course, a large and growing literature on how various reprocessing schemes currently work and will work in the future. However, for systems modeling purposes we again appeal to a “black-box” outlook in which we care mostly about the composition vectors of

the output materials that emerge from the box. Thus, we implement the common notion of a matrix-based separation transformation that places some given fraction of each element from the input stream into each of some set number of output streams, as shown in Figure 3.4. Note that, for each element, the sum of the stream-wise separation coefficients must sum to one (which enforces conservation of mass), and that each isotope of a given element is treated the same (since we're representing *chemical* processes, not physical ones).

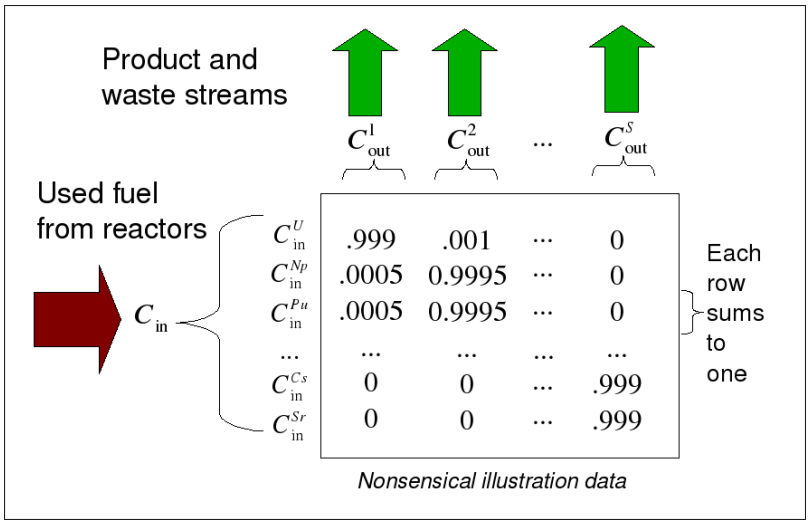

Figure 3.4 Conceptual sketch of the reprocessing scheme currently implemented in the separations class's main operation, which divides the material from incoming used fuel into multiple outgoing product and waste streams. The separations matrix data are stored as `ChemBook` data structures that each represent the chemistry of one real-world reprocessing scheme (see Table 3.6).

For now, GENIUSv2 supports four product streams—one for uranium; one for plutonium and any actinides designed, for non-proliferation purposes, to travel with it; and two available for arbitrary combinations of minor actinides. These streams were chosen to maximize the number of real-world reprocessing schemes the model could be adapted to represent and to ensure that the streams would be useful for constructing a variety of recycled fuel recipes. However, at least as GENIUSv2 currently operates, it would not be disruptive to change this design, since fabricators wishing to purchase recycled material do not order from these separated streams directly (see Section 4.4).

The data necessary to represent the matrix shown in Figure 3.4 are stored in complex sparse data structures called `ChemBook`s. At least as currently proposed, separations plants are likely to be very large and will probably be expected to adapt to several different separations schemes. Thus, rather than assigning separations plants a single `ChemBook`, we store the list of all the possibilities (some of which are included in the code and others of which users will be able to provide) as a static[11] data member in the separations class. This design provides all separations plants in a simulation with access to the complete data; as research and testing progress, developers can implement an appropriate mechanism for helping individual plants choose between the various options at a given time. This static member and other significant data and functions for the separations class are shown in Table 3.6.

Table 3.6 Selections from the separations class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	**Description**
[Inherited data]	See superclass (facility) data, Table 3.1.
`StreamInventory separatedStreams`	A buffer of material objects sorted according to the product stream each object emerged from reprocessing as a part of.
`static ChemLib processData`	A collection of `ChemBook` objects storing the data that describe available separation schemes.

Member functions

Declaration	**Description**
[Inherited functions]	See superclass (facility) functions, Table 3.1.
`void separate()`	Performs a month's worth of reprocessing on material from this plant's stocks, according to the current separation scheme.

Another outstanding research question for GENIUS developers will be how handle the discrete nature of the materials moving between a facility's buffers and its process lines. In other words, though we can think of a discrete material object arriving at a facility as representing a single shipment or lot of material, we need to figure out what should happen as (or perhaps if) these materials get combined together for processing. Although it probably applies to conversion, enrichment, and fabrication plants as well, this question of material fungibility takes

[11] See Appendix A.

on added significance in the separations context. Batches of used fuel arriving from different reactors (and possibly after different irradiation and storage times) comprise the variety of the vintner's selection of grapes in Piet's winery analogy (2007) discussed in Section 2.2.1. If we completely homogenize all material in the black box, we diminish some of that variety. If we do not homogenize at all, we risk artificially augmenting the variety by modeling separations as a heavily batched process instead of a continuous one. There are computational concerns as well, since the number of discrete separated material objects we can choose from to fill orders for recycled fuel materials determines the size of the linear program we use to formulate the recipe approximation problem (see Section 4.4). For now, we choose a *via media* and effectively homogenize feed stocks as they enter each month. Thus, each month we turn one large material object (formed from all the used fuel we've received since last month) into S separated objects, representing the total mass flow of each of the S separation streams if *only* the material from our single object moved through the system.

3.1.4.3 The repository class

The GENIUSv2 repository class is not yet very sophisticated, but we discuss it in some detail here because it has the potential to robustly and flexibly support many types of future analysis. At present, we treat repositories according to the simplistic mass-based load limit model. Thus, the items of interest in the class definition (see Table 3.7) relate to monitoring and limiting the mass of material that enters.

The repository class is the only one whose `capacity` member represents a cumulative rather than monthly limit. A convenient side effect of mass-based load limits is that they allow the code to make decisions about future loading based on a single scalar quantity: the capacity less the mass that has entered so far. If we're also willing to assume that the repository is non-retrievable, then once we've incremented `massIn` appropriately, we no longer care about the materials that have entered the repository, and we can relieve ourselves of the burden of storing them in memory for the remainder of the simulation. Thus, in the current implementation, the

`dumpMat()` method records the mass of newly arrived materials each month and then deletes them[12].

Table 3.7 Selections from the repository class definition. Function arguments and some members omitted for space or complexity reasons.

Member data

Declaration	**Description**
[Inherited data]	See superclass (facility) data, Table 3.1.
`double capacity`	In the case of repositories, the *cumulative* limit on the mass of material this facility can accommodate.
`double massIn`	The mass of material that has been emplaced in this repository.

Member functions

Declaration	**Description**
[Inherited functions]	See superclass (facility) functions, Table 3.1.
`void dumpMat()`	Deletes from memory all the materials currently in the `stocks`.

The current implementation does not preclude studies of how repository heat and radiotoxicity loads change over time; as long as the code stores a record of each material's composition at emplacement, the sum of all materials' contribution to these loads can be reconstructed as a post-processing step, if desired. However, if we wanted to enforce other repository load limits dynamically, the repository class would need to change somewhat. An obvious and useful extension of current capabilities that would still allow `dumpMat()` to delete emplaced materials outright would be to adopt Radel's methodology (2007) for dynamically converting material composition to a characteristic repository length, based on the three temperature limits that also constrain allowable loadings. However, her work and the limits themselves were based on Yucca Mountain-specific heat transfer calculations and would not be directly applicable for modeling repositories at other sites.

Many possibilities for retrievable repository storage, dynamic load limiting, and other repository modeling problems of interest become possible if we are willing to sacrifice *some* discrete-materials data and devise a means of modeling the material in the repository in a more

[12]The material class destructor (the function that gets called to delete a material object) is written such that it sends its history data to the bookkeeper just before deletion. Thus, deleting materials does not mean we lose their associated records (see Section 3.2.2)

memory-efficient manner than simply storing millions of discrete material objects in memory. For instance, one can imagine that it might be desirable to incorporate some sense of the geometry of a repository by modeling the contents of each drift (tunnel) in some aggregated way.

3.2 Simulation machinery

Having thus described the component parts of the GENIUSv2 fuel cycle model, we move now to the mechanisms by which we help those actors work together in a coordinated simulation. Pidd organizes his discussion of this aspect of codes like GENIUS (which are known in the management science and operations research literature as *discrete event simulations*) in terms of a *simulation clock* that moves time forward, an *executive* that oversees entities' cooperation, and an *event list* that records what happened (Pidd, 2003, p. 239-241). In GENIUSv2, these functions are handled by the timer, manager, and bookkeeper, respectively.

3.2.1 Timer

Like it's predecessor, GENIUSv2 uses a one-month time step, which has been deemed appropriate for capturing the appropriate level of detail. It might at first seem as if a three- or even six-month time step would be sufficiently small, since real-world reactors operate on 12-, 18-, or 24-month cycles in order to time their outages with the spring and fall nadirs in seasonal energy demand curves. However, in the DF/DM paradigm where we're interested, in part, in fuel cycle supply chain robustness, it seems wise to divide time more finely. By choosing a single month, we can model delays on roughly the order of the length of a reactor refueling and maintenance outage. The GENIUSv2 defaults are for a 1200-month simulation that begins in January, 2010, but non-standard durations and start years can be given as command-line arguments. The code converts all dates given in the input file to a system where each month is represented by an integer, with $t = 0$ representing the start month.

Once the code has completed reading of the input file and construction of the model specified therein, the timer begins passing time-step messages to the simulation objects, two per

month. The mechanism for this advancement is as follows: Each simulation entity (regions, institutions, and facilities) and the manager implement methods called `handleTick()` and `handleTock()`, each of which takes the single argument of an integer that gives the current time. The timer invokes this method on the manager, which in turn invokes it on all the regions, which in turn invoke it on their member institutions, etc. Most of the important business of a time step is handled during the tick phase. For instance, if an institution's plan for building new facilities indicates that it should begin construction on one this month, it instantiates one and adds it to its collection. Similarly, if it's time for a facility to begin a new operating cycle, its `handleTick()` function will invoke its `beginCycle()` function, which might set in motion any number of actions including making requests and offers for material and performing characteristic operations on the material it already has. Because the first task each entity performs during the tick phase is to pass the tick on to whatever entities it's responsible for, the final entity to actually handle its assigned jobs is the manager itself. Thus, by the end of the tick phase, the manager is able to "take stock" of the system as a whole; it has received reports from all the entities that currently want to be matched with a supplier or customer.

The first action that happens after the manager's `handleTick()` method returns and the timer issues the tock is that the manager performs an algorithm for deciding on the set of actions the system should take and issuing the corresponding instructions—usually by invoking the `executeOrder()` functions on the suppliers that it has matched to customers. Once these instructions have all been issued and followed (that is, all material transactions have been completed), the tock gets passed down the R-I-F hierarchy, and each entity performs the internal bookkeeping tasks necessary to record what it did this month. The main such task is recording an entry in an internal capacity factor log; for instance (using our previous notation), if a conversion facility with a monthly capacity of Z_{month} tons of uranium hexafluoride actually converted only Z tons, then it records a capacity factor of $\frac{Z}{Z_{month}}$.

The timer class, as well as the manager and bookkeeper, are implemented as singleton[13] classes. The timer is an especially useful object to implement in this way, because then any

[13] See Appendix A

object that needs to check the time—from within a function that doesn't take the time as an argument, that is—can do so without having to store a reference to the timer as a data member.

3.2.2 Bookkeeper

The bookkeeper class, tasked with recording the state of the simulation in an output file, also takes good advantage of the features of the singleton design pattern; rather than forcing the bookkeeper to somehow keep tabs on every single object in the simulation, we make sure that each object can instead get in touch with the bookkeeper, notifying it of the important events that need to be recorded. There's no need to do so immediately after those events, though. Facilities and materials keep track of their own histories until they're deleted—either at the end of the simulation or when the code decides they no longer need to be tracked. Thus, when a facility's destructor is called, the last thing it does is call the bookkeeper's `writeFac()` method, which collects and writes out the information that needs to be included in the output file (see Section 3.3). A similar procedure applies for writing out material location and composition histories, and also those of the batches (if batch- rather than assembly-wise tracking has been selected). Another attractive feature of this design is that the bookkeeper serves as a "wrapper" around the interface that gets used to write the output file, appropriately encapsulating the code that depends on that interface. Thus, if future development calls for the output file format to change, only the code within the bookkeeper class needs to be rewritten.

3.2.3 Manager

The manager has the most difficult job of any class in GENIUSv2. I gestured toward this job in the description of the timer: During the tick phase, the manager collects information about the needs of various entities throughout the model. At the beginning of the tock phase, its `match()` method attempts to identify the optimal routing for meeting those needs in accordance with the objectives of the fuel cycle system as a whole. Chapter 4 presents

optimization formulations for making these decisions, but because those methods are encapsulated elsewhere, we're in a position now to describe the overall procedures for the manager's data-collection and instruction-issuing tasks.

Figure 3.5, an extension of the fuel cycle model shown in Figure 3.1, illustrates how the members of the R-I-F hierarchy participate in this procedure. The mechanism for that participation is the passing of *messages* from entity to entity. When a facility has a product or service to offer, or a request for the product or service of another facility, it constructs an instantiation of the message class and encodes the details of its offer or request therein. The important data included in a message are the commodity being offered or requested, the amount of that commodity available or required (a sign convention specifies whether the amount represents an offer or request), and information about the entity sending the message.

Although, as a singleton class, the manager is directly reachable by all simulation objects, we instead use the convention that messages get passed up the hierarchy and reach the manager after being examined by the originating facility's institution and region. We made this decision in the hope that as GENIUSv2's R-I-F hierarchy becomes richer and more meaningful, institutions or regions might modify messages based on their own "wider view" of the state of the hierarchy below them, perhaps even taking an active role in matching customers to suppliers. For instance, a vertically integrated institution that owns several different kinds of fuel cycle facilities is less likely to seek outside suppliers of commodities it is capable of producing itself. Thus, it's conceivable that at some time we may wish for institutions to intercept and handle messages that are requests or offers for in-house commodities from the perspective of that institution. The same methodology might be applied to regions. These methods might be advantageous from a modeling-realism perspective, but it also seems likely that they could harm the system's ability to seek out globally optimal solutions; developers will need to take great care in determining how to use them appropriately if and when they are implemented. For now, we merely note that, in accordance with GENIUSv2's flexible design philosophy, the capability for this message interception technique is present in the model (at negligible computational cost) and can be used if deemed appropriate.

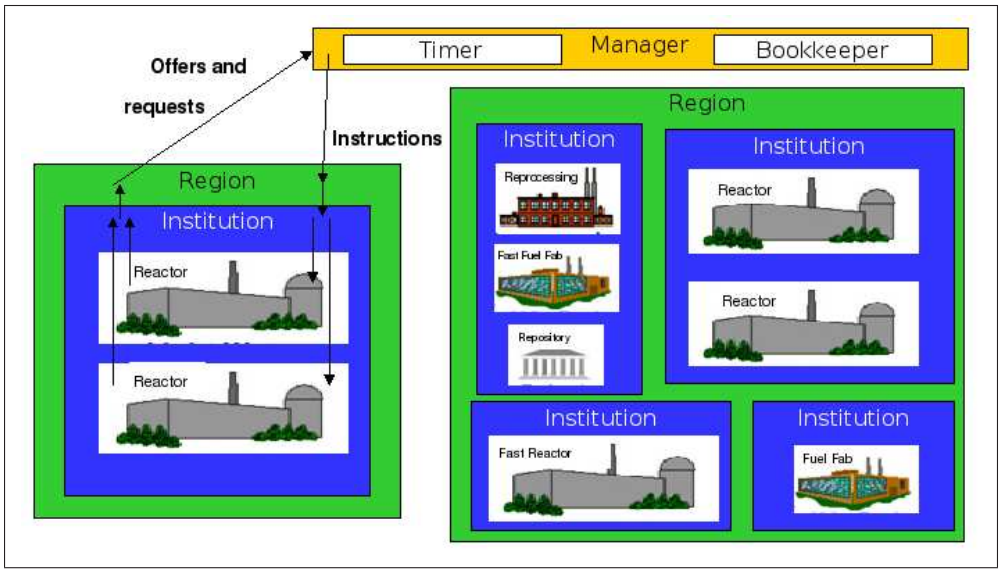

Figure 3.5 Messages representing offers and requests get passed up the hierarchy for matching during each time step. Future implementations may allow institutions or regions to handle some matchings themselves, when appropriate. After the manager performs the matching, it issues corresponding instructions to the suppliers. Image adapted from Wilson and Oliver (2007).

For now, though, all messages reach the manager. As they arrive, the manager sorts and stores them by commodity and by whether they're offers or requests. When it starts to match them at the beginning of the tock phase, it does so in a very specific order. The need for the order is related to a supply chain challenge caused by the aforementioned problem of material fungibility.

Some commodities, like yellowcake and unenriched uranium hexafluoride, are highly fungible; they have a predictable composition, and equal amounts of these commodities can be exchanged for one another without adverse consequences. Thus, unless we have a strong desire to study the effects of variability of uranium ore quality and therefore to vary the composition of yellowcakes from one mine to another, mines can make reliable and unambiguous material offers by simply calculating the mass of yellowcake they have on hand and only offering that amount. Enrichment plants, on the other hand, cannot make such unambiguous offers of material. The commodity they deal in is less fungible because value is added continuously to the uranium hexafluoride as it is enriched to higher and higher U-235 mass fractions; no one

would trade equal masses of uranium one for another if they had different enrichments. Thus, without knowing something about the enrichment needs of fuel fabricators, there is no way for an enrichment facility to offer a meaningful quantity of material, as product; it doesn't know how much that product will need to be enriched.

Instead, we let suppliers of less fungible material offer *services* rather than *product*. An enricher's offer then specifies not the quantity of material it can make, which is under-determined, but the amount of enrichment it can perform, which is fully determined by the plant's capacity less any outstanding commitments. Buyers of less fungible material, on the other hand, specify the exact composition of the material they want—a total mass and an enrichment, in the case of uranium hexafluoride. The manager then bears the burden of translating the quantity given, in either the offer or the request, to reach a common unit that can serve as the basis for determining supplies and demands and finding a desirable matching.

This matching strategy has the following consequence: suppliers of front-end fuel cycle *services* (enrichers and fabricators) may need to order feed material *after* being matched to a customer. As an illustration, note that in order to minimize the logistical lag time that can result from a reactor ordering enriched uranium oxide fuel, the manager matches the relevant commodities in this order: (1) fuel requests with fabrication offers, (2) enriched uranium hexafluoride requests with enrichment offers, and (3) unenriched uranium hexafluoride requests with offers of same (the complete order is given in Table 3.8). This way, if a fabricator does not have suitable material on hand to construct a fuel batch ordered by a reactor, it immediately orders the enriched material it needs, and that order arrives at the manager *before* the manager matches fabricators to enrichers. A similar procedure occurs if the enricher does not have sufficient and suitable material to enrich. Any unmatched orders for these non-fungible material are stored by the manager, which re-attempts to match them during the following time step.

The order of commodity-wise matching is just one of a number of problems that we might label fuel-cycle "tuning." The goal of such tuning is to ensure that delays in material availability result only from cases where there is a legitimate real-world bottleneck or material

Table 3.8 Current manager order for matching fuel cycle commodities.

Enumeration	**Full name**
`eUoxFuel`	Enriched uranium oxide (LWR) fuel
`uUoxFuel`	Unenriched uranium oxide (PHWR) fuel
`moxFuel`	Mixed oxide (recycled) fuel
`eUF6`	Enriched uranium hexafluoride
`uUF6`	Unenriched uranium oxide fuel
`cake`	Yellowcake (uranium ore, mostly U_3O_8)
`usedFuel`	Used fuel from all reactor types

shortfall, not from places where the behavior of the simulation entities or manager is insufficiently sophisticated to deliver the necessary material when it is theoretically possible to do so. Admittedly, tuning the fuel cycle's behavior involves building in various decision heuristics about when and how facilities should offer or request goods and services. I believe the long-term goal of these tuning activities should be to identify and build in whatever decisions are an absolute given in the real-world and to parameterize the rest, requiring these parameters to be user-specified or part of the future optimization engine's parameter space. For instance, if we were to discover that the fuel cycle performs best when enrichers and fabricators maintain backup stocks of raw materials (perhaps guessing the specifications of future fuel orders based on the distribution of past orders), then the exact amount (and in the case of the fabricator, the enrichment) of backup material these facilities should attempt to maintain might be an input parameter worth exploring.

I should be clear that, as currently implemented, the set of default facility and manager behaviors that together determine the system behavior of possible GENIUSv2 fuel cycles are *not* very well tuned. In order to simply get all of the basic features and functions desired for GENIUSv2 in place (and to do so in a relatively straightforward way), I have tried to make the simplest possible (and therefore often naive) choices about how facilities and the manager behave. This is a necessary starting point for the more sophisticated modeling that can now evolve as developers tune the structures that are in place.

3.2.4 Other classes

A few minor support classes exist in GENIUSv2 and have not yet been discussed. The two most prominent are the solver wrapper and the input reader, which serve analogous roles to the bookkeeper from an encapsulation perspective. Just as the bookkeeper is a GENIUS-friendly interface to the software tool used for recording output files, so is the solver wrapper a simplified interface to the external solver technology GENIUS uses for optimization problems. This class will be discussed in the next chapter. Similarly, the input reader wraps around and handles the parsing of the input file, constructing the simulation's initial condition as it does so. I discuss it below.

3.3 Input/output infrastructure

As should now be obvious, a large and highly structured volume of data must be passed in to GENIUS to describe a meaningful fuel-cycle scenario, and an even more challenging data set needs to be returned by the code in a way amenable to efficient post-processing. Because nuclear fuel cycle modeling is a fairly immature field (compared to core physics modeling and other more physical problems that nuclear engineers have studied for years), there are not yet any standardized text or binary file representations for fuel cycle input or output data. While the GENIUS-specific model we present here is unlikely to fill that role, my hope is that it can serve as a possible model for how to take advantage of modern scientific computing resources to robustly populate and visualize fuel cycle systems models.

One natural way to capture both the hierarchical structure of the data needed for GENIUS input (descriptions of the various regions, institutions, and facilities) and the transactional structure of its most important output data (material location histories) is via a relational database methodology. The popular SQLite database (SQLite Consortium, 2008) is a natural choice because it requires little overhead; is highly stable, standardized, and well tested; and is freely available to anyone as open-source software. Moreover, it is supported via a built-in interface by the Python high-level programming language (Python Software Foundation, 2006). Python

is another well supported and highly popular open-source computing tool with extensive libraries for numerical and scientific computing. Myself and other members of the GENIUSv2 development team have written and maintain adaptable pre- and post-processors that I expect will continue to improve GENIUSv2 ease of use. These processing libraries themselves are somewhat beyond the scope of this paper, but the design of the input and output file format itself is not. For more information about actually constructing these SQLite databases, including some limited efforts to extract Juchau's hard-coded GENIUSv1 input data into a GENIUSv2-compatible database, contact CNERG at `http://cnerg.engr.wisc.edu`.

3.3.1 Scenario specification

Conceptually, creating a GENIUSv2 input file requires specifying (1) all the regions, institutions, and facilities in the model at the beginning of the simulation (i.e., an initial condition), (2) the static plans for when each institution should build future facilities, (3) the parameters of these generic future facilities, and (4) a set of rules describing non-standard relationships between members of the R-I-F hierarchy, if desired (see Section 4.2). The first two tasks are accomplished by populating database tables called `Regions`, `Insts`, and `Facs`; the third, by populating table `FacParams`; and the fourth, by populating `Rules`. While these five tables are likely to persist indefinitely, it has unfortunately been my experience that it is very hard to stabilize the set of required columns in each table. When GENIUSv2 "goes public," it will be important for the development team to publish clear instructions about which input parameters are required for running a simulation with a given stable release. For instance, whenever developers add a new modeling parameter that needs to be specified by the user, code must be added to read this parameter from the appropriate column—code that will likely need to complain if a given input file does not contain that column.

A current listing of each table's required columns is given in Appendix B, but a glance at an actual example should sufficiently illustrate the overall approach. Figure 3.6 shows screen shots of several tables from a fairly simple input file[14]. The most intuitive is the first, the `Facs`

[14] It's actually the real input file used for one of the demonstration problems (see Section 5.1.2).

table. We can see from the number of rows that this scenario includes six initial facilities, and the fourth and fifth columns (`yearStartOp` and `monthStartOp`) tell us that all of them start operation in January 1970. The second-to-last visible column (`type`) shows that they are all front-end fuel cycle facilities (mine/mills, conversion, enrichment, and fabrication), and the fact that each will exist 1560 months (see Column 7, `lifeTime`) suggests this simulation is not designed to model the detailed comings and goings of fuel cycle facilities with more realistic lifetimes. Clearly, this is a simplified and idealized demonstration problem, at least from the perspective of the fuel cycle support facilities.

But where are the reactors? Unfortunately, it's difficult to tell for this problem, in which I've chosen not to fully describe any "historical" reactors in the `Facs` table. Reactors will be built; we just can't see where they are in the tables. While a database is appealing for displaying and perhaps even entering input information that is representable as text or as scalar values, vector quantities present a slight complication, at least at first glance. But it turns out that SQLite supports a data type called a blob, which is a simple chunk of binary data. GENIUSv2 and its pre- and post-processors all acknowledge a data convention for storing vectors of either integers or double-precision values as a blob. These blobs can be stored in a single database column, but they can't be rendered by the database browser because, to it, blobs are just raw binary data. The column labeled `build` in the `Insts` table actually encodes complete sets of instructions for how each institution should build certain future facilities at certain times. We *can* actually tell what each of these future facilities will look like by examining the `FacParams` table. Since there's only one row defined, we know that all facilities built during the the simulation will be pressurized water reactors (we've named this model "Generic PWR") with the parameters given in the subsequent columns.

An input reader object performs the tedious but straightforward task of reading through each table and instantiating the various GENIUSv2 objects and data structures necessary to build the model described in the database. Because of the relative difficulties of writing C++ and Python code, and especially working with each language's interface for reading SQLite databases, we let the input reader assume that the given input file is valid, leaving the seemingly

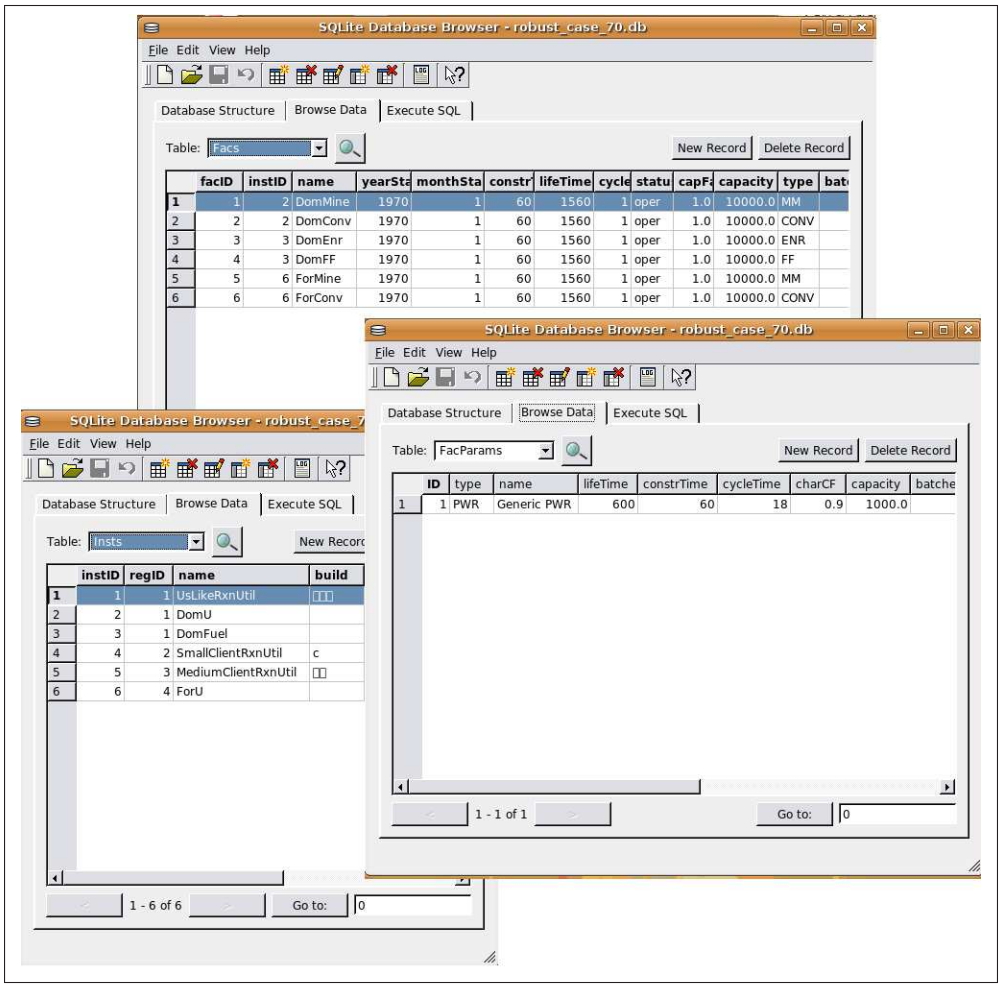

Figure 3.6 Partial views of a three tables from a sample GENIUSv2 input file. Note that vector quantities can be stored as blobs that can be read by the code but not rendered in this database browser. See Appendix B for a complete description of each table and its columns.

endless task of input file validation to the pre-processor. Once the input reader has read all the tables and created all the objects, the application instantiates a timer that can handle a simulation of the length and start time passed to the code as command-line arguments. The simulation then begins.

3.3.2 History recording

The output file for GENIUSv2 is a copy of the input database to which the bookkeeper adds extra tables and extra columns in the existing tables. As discussed in Section 3.2.2, the operational history of facilities (the GENIUS time at which they began operating and their actual capacity factors, by month, thereafter) and the location and composition histories of materials gets sent to the bookkeeper just before these objects are deleted from memory. Recording the facility output is very straightforward; for facilities that were explicitly specified in the original input file (i.e., not part of a future build plan), we only need to add the history information to their entry, which already exists. These data go into special `startOp` and `capLog` columns that GENIUS adds to the `Facs` table. The latter column is again a specially formatted blob representing a vector of doubles.

Material histories get spread across the two tables shown in Figure 3.7, `MatFacHist` and `MatIsoHist`. Unsurprisingly, each of the transactions that the material recorded about itself during the simulation shows up as a row in `MatFacHist`. Each row records the ID number of the material that was transfered, the time the transfer took place, the integer identifiers of the source and destination facility (the `facID`s from the `Facs` table, of course), and a composition identifier that links this row in `MatFacHist` to the row in `MatIsoHist` that stores the isotopic composition at that particular time (another specially formatted blob). For those times when a material was "decayed on demand," the entry in `MatFacHist` contains duplicate entries in the `fromFac` and `toFac` columns.

The navigation buttons near the bottom of each screen shot in Figure 3.7 show the number of individual records in each table. Thus, we can see that the 770 facilities in this simulation (764 of them reactors) created and used materials that underwent 55,850 discrete facility-transfer or decay events and exhibited 92,260 isotopic states that were recorded. The total output file size was 11.6 megabytes.

A couple of concluding remarks are called for here. First, note that, for the kinds of early demonstration problems I discuss in this thesis, the data in these large output files is hugely redundant. The 700+ identical reactors all eject identical batches of material characteristic of a

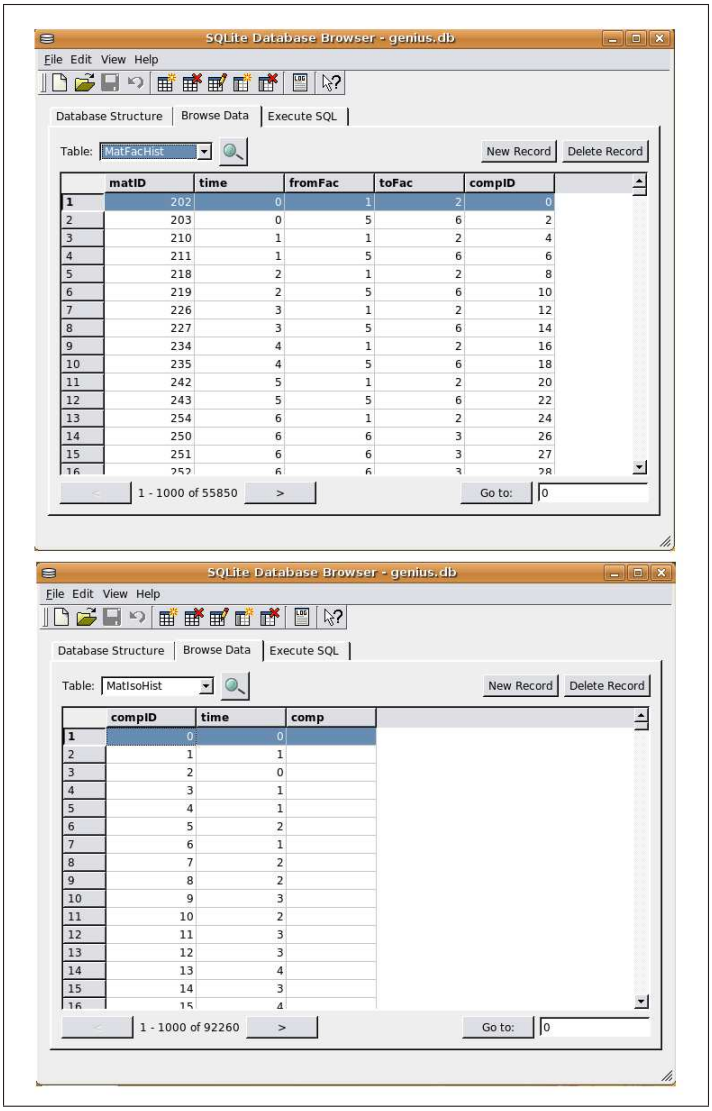

Figure 3.7 Partial views of the two material history tables from a sample GENIUSv2 output file. We can observe high level of detail in DF/DM modeling by noting the large number of records in each table. Recall that blob data (`MatIsoHist`'s `comp` column) cannot be rendered by this browser but is present.

particular burnup value implied by the input and output recipes assigned to the single generic reactor type. And the same enrichment plant and fabrication facility work together to fabricate that same input recipe over, and over, and over. The important thing to remember is that what looks like massively inefficient overkill now is appropriately robust for larger, more detailed and realistic problems, especially when the code is capable of simulating arbitrary burnups.

Although writing to the database is a slow operation, the highly detailed records of simulation history data that this output design can accommodate is the very *raison d'être* of DF/DM codes.

Second, let me also mention that this tabular scheme for material histories is a very compact representation of a huge 4-D dataset whose axes are the time, the originating facilities, the destination facilities, and the masses or number densities of each isotope. Although doing so sometimes requires clever pre-collapsing of the dataset at the time the database is queried, it is highly convenient to construct this 4-D array (or subsets of it) when examining GENIUS output data. Thus, we see that another advantage of the SQLite/Python work flow is the existence of robust scientific and numerical tools for data manipulation, making it comparatively simple to add sophisticated data analysis and visualization capabilities to the GENIUSv2 post-processor. This ease stands to benefit end users interested in standard fuel cycle visualizations but uncomfortable with the tricky task of extracting useful information from so large a data set.

3.4 Summary

This chapter described the design and implementation of GENIUSv2, including its model of the nuclear fuel cycle, the support machinery that help the simulation objects work together and that record the results of this cooperation, and the infrastructure for passing input to and receiving output from the code itself. Central to the functioning of this design is the manager, which receives messages regarding the state of supply and demand in the fuel cycle system and issues instructions for matching customers to suppliers. What remains to be explained in Chapter 4 are the mathematical formulations that govern these matching processes.

Chapter 4

Optimization formulations

Chapter 3 described the novel modeling capabilities and robust software design of GENIUSv2. This chapter discusses the other main contribution of this work, formulations for some of the optimization problems either raised by or made more important by the DF/DM modeling paradigm.

Without venturing a precise formulation, we note first that the purpose or objective of a global nuclear fuel cycle from an optimization perspective is to come as close as possible to meeting the electricity needs of each region for as low a total cost as possible. Of course, a tremendous amount of work is needed to make that definition precise. For instance, we might ask if the objective function should penalize overproduction of electricity, or how to factor in costs associated with, say, nuclear waste disposal. But, in theory, if we believe that we can turn the various aspects of fuel cycle system performance into costs, then some kind of cost-minimizing optimization approach ought to be effective in identifying promising nuclear fuel cycle designs.

However, implicit in our decision to try to tackle this optimization problem via discrete-event simulation[1] is the acknowledgement that an explicit form for fuel cycle system optimization would be very challenging to identify, let alone solve. The system is highly complex, comprised as it is of differentiated components (facilities) that depend elaborately on one another, many of them operating nonlinearly on the materials they process. Merely identifying a

[1]Of course, the decision to make GENIUS a discrete event simulation was not based wholly (and perhaps not even in part) on optimization considerations. Indeed, the desire to perform direct simulation and to analyze DF/DM data are the prime motivators and optimization a secondary concern.

self-consistent set of constraints to describe how the system as a whole operates (to say nothing of constraints on how individual facilities may be deployed) is a daunting and quite possibly fruitless undertaking. In a review of supply chain optimization problems similar to our nuclear fuel cycle context, Vidal and Goetschalckx focus on mixed-integer program formulations (Vidal and Goetschalckx, 1997). These problems are NP-hard in general (Gray et al., 1997), and even techniques for solving well studied mixed integer programs with important applications are perhaps beyond the level of mathematical sophistication and computational expense appropriate for a code like GENIUS, at least for the time being.

Instead, we make a series of simplifying assumptions, the first of which suggests the framework illustrated in Figure 4.1. We assume that, for a *given* fuel cycle design (including a facility deployment plan and perhaps values of other important parameters describing those facilities and how they work together), there exists some optimal plan for routing materials through the system over the duration of the simulation. Such a plan will minimize the cost of operating the fuel cycle, including possible contributions from the penalizing cost of under-producing the specified amount of electricity in each region. The ultimate task of the simulation manager's matching efforts is to solve this materials routing problem (MRP). In fact, it's not unreasonable to think of the course of normal GENIUSv2 operation as identifying, simulating, and recording the material flows and facility operation histories that correspond to an MRP solution for a given fuel cycle scenario. Future developers can then attempt to solve the outer fuel cycle design problem (FDP) iteratively via optimization toolkits that search the input decision space[2]. Most of the remaining simplifications we will make relate to how we solve the MRP itself and are discussed in Section 4.2.

[2]This nested "division of computational labor" is not unlike familiar approaches to other optimization problems of interest to nuclear engineers, especially pin placement in reactor core engineering. The main difference is that, in this context, the "inner simulation" itself also includes an optimization problem (as opposed to the core physics radiation transport problems solved repeatedly in that more typical example).

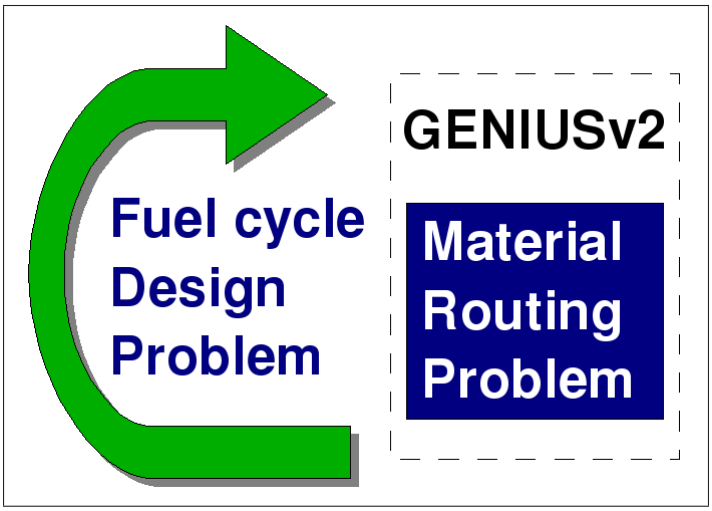

Figure 4.1 Decomposition of the task of fuel cycle optimization into a design problem to be solved iteratively (and externally) and a routing problem to be solved during the simulation.

4.1 Linear and network flow programming

We choose as our optimization strategy for solving the MRP a set of formulations from network flow program (NFP) theory. Network optimization problems adopt fairly naturally to our fuel cycle context because they too are concerned with the flow of material from sources (in our case, suppliers) to sinks (customers). This section serves as an introduction to NFPs, since they are likely unfamiliar to a nuclear engineering audience. However, because network programs are a special case of more general optimization problems known as linear programs (LPs), and because we use these more general methods in tackling another fuel cycle optimization problem, we first introduce them.

A linear program is an optimization problem where we wish to minimize or maximize a linear *objective function* of some vector, subject to a set of linear constraint equations on its components. Using the LP notation of Ferris, Mangasarian, and Wright (2008), we first define

the variables x as the following column vector of non-negative components:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_l \end{bmatrix}$$

We can then define a constraint-equation coefficient matrix, A, a constraint-equation right-hand-side vector, b, and a set of objective-function coefficients, p, like so:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1l} \\ \vdots & \ddots & \vdots \\ A_{k1} & \dots & A_{kl} \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}, \qquad p = \begin{bmatrix} p_1 \\ \vdots \\ p_l \end{bmatrix}$$

If we call the objective function value z, then one possible way to state the LP is

$$\begin{aligned} &\min_x && z = p^T x \\ &\text{subject to} && Ax \leq b,\ x \geq 0 \end{aligned} \tag{4.1}$$

where the superscript T indicates the transpose operator. Ferris and colleagues note that problems with any linear objective function subject to a system of linear constraints (including ones where the x components are not necessarily non-negative or the constraints are given as strict equalities or some combination of the "$\leq$, " "$\geq$, " or "$=$" operators) can be expressed in this form via a series of standard transformations (2008, p. 9).

Not all linear programs have an optimal solution. First, a problem can be *infeasible*, which happens when the region of N-space that satisfies all the constraints, the so-called *feasible region*, is null. Sometimes infeasibility is easy to spot, as in when separate inconsistent constraints exist on a single variable (for instance, $x_2 \geq 0$ and $x_2 \leq -3$). But for large constraint sets of many variables, infeasibilities can easily escape one's notice, so it's important that solution algorithms know how to check for them along the way. Second, an LP can be *unbounded*, which means the feasible region is infinite along a direction that causes the objective function to increase or decrease (for maximization and minimization problems, respectively), which means z can be arbitrary large or small and hence no optimal value exists. Again, solution algorithms must recognize this behavior.

The possibility of a highly simplified sketch like this one notwithstanding, LPs are well studied problems with a rich theoretical underpinning and robust solution algorithms for large problems. These algorithms include the classic *simplex method* and a number of newer and sometimes faster methods and are implemented in a wide range of solver software. The proprietary CPLEX solver (ILOG, 2008) is the gold-standard, but well maintained and high-performing open-source solvers exist as well. These include the COIN-OR project's CLP library (2007), which GENIUSv2 currently uses to solve LPs.

The generality of Equation 4.1 hints at the ease with which LPs can be adapted to a variety of specialized optimization and modeling contexts. Often, these more specific formulations have given rise to sophisticated literatures of their own and solution methods with greatly improved performance. Linear network flow programs are one such example, a sort of linear programming-meets-graph theory subfield useful for modeling problems where fluid, data, goods, traffic, or decision-logic flow from *source nodes* to *sink nodes* through a network of *arcs* (and possibly some *transshipment nodes* that neither produce nor remove flow). Figure 4.2 shows a simple four-node, four-arc network with two sources and two sinks.

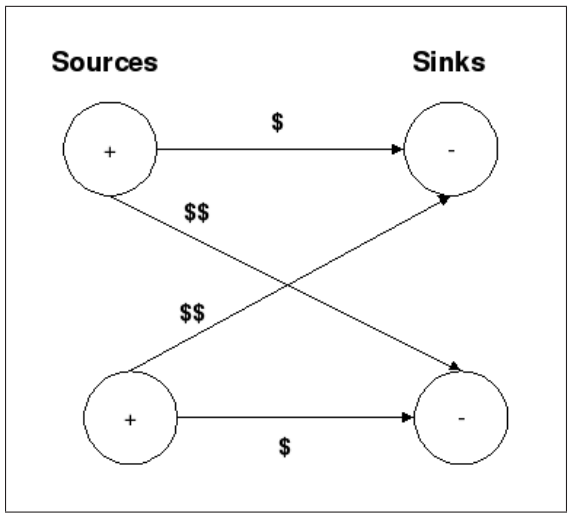

Figure 4.2 A very simple linear network with two sources and two sinks. Unit flow on the diagonal arcs is more expensive than on the horizontal ones.

Using the notation from Bertsekas (1998), we describe a linear network program in terms of its node set N and its arc set A. Each node $i \in N$ has an associated *divergence*, s_i, a

net in- or out-flow that is positive for sources, negative for sinks, and zero for transshipment nodes. Each arc in A has a unit flow cost, a_{ij}, and a range of valid flow values, $[b_{ij}, c_{ij}]$. Thus, the constraints for this NFP are that the flow x_{ij} on each arc (i, j) is within its flow bounds (Equation 4.2) and that the divergence constraint at each node is satisfied (Equation 4.3):

$$x_{ij} \in [b_{ij}, c_{ij}] \forall (i, j) \in A \tag{4.2}$$

$$\sum_{j|(i,j)\in A} x_{ij} - \sum_{j|(j,i)\in A} x_{ji} = s_i, \forall i \in N \tag{4.3}$$

(Note immediately that $\sum_{i\in N} s_i = 0$ is a necessary condition for feasibility; this is an important consideration when modeling practical problems in which there is no guarantee that supply equals demand.) The objective in a minimum cost network flow problem is to satisfy the flow constraints while minimizing the total cost of the flow, $\sum_{(i,j)\in A} a_{ij}x_{ij}$. Thus, the complete formulation is given by Equation 4.4:

$$\begin{aligned} \min_x \quad & \sum_{(i,j)\in A} a_{ij}x_{ij} \\ \text{subject to} \quad & x_{ij} \in [b_{ij}, c_{ij}],\ \forall (i, j) \in A \\ & \sum_{j|(i,j)\in A} x_{ij} - \sum_{j|(j,i)\in A} x_{ji} = s_i, \forall i \in N \end{aligned} \tag{4.4}$$

Careful examination of Equation 4.4 reveals that it is indeed a special case of Equation 4.1. However, it turns out that the matrix A in Equation 4.1 has a special structure for network flow programs, a structure that can be exploited to greatly simplify the solution methods for these problems. Thus, they can be solved via the *network* simplex method or other specialized algorithms, which Chinneck reports can be hundreds of times faster than the normal LP simplex method (2007). The CLP library includes mechanisms for taking partial advantage of network problems' special structure, but it does not implement a full-fledged network simplex method (Forrest et al., 2004). However, the network programs we currently solve in GENIUS are not unreasonably expensive even when solved with standard LP technology.

4.2 Materials routing problem: Formulation

While it is easy to intuit the usefulness of network formulations for solving the GENIUSv2 MRP, two major difficulties bar us from modeling the nuclear fuel cycle with Equation 4.4 directly. The first is that this standard form is meant for the so-called *single-commodity* problem. All the flows x_{ij} must be in some sense homogeneous; a single cost and pair of flow bounds must apply to all the flow on a single arc, and the divergences of sources and sinks must measure the production and removal of the same basic material. It seems unlikely that a useful formulation exists for modeling the nuclear fuel cycle with a single-commodity network flow problem; even if we overlook mc^2 losses in the reactors and pretend that total mass is conserved (which would allow the flow conservation constraint in Equation 4.4 to apply throughout the network), the fuel cycle materials flowing through the various sectors of the fuel cycle are too heterogeneous to expect a single-commodity, "one-mass-fits-all" formulation to be useful. We simply care too much about the differences between materials in different parts of the fuel cycle (their chemical and physical form, their isotopic composition, enrichment and criticality profiles, etc.) to ignore their differences, and we must therefore treat yellowcake, enriched uranium hexafluoride, fabricated fuel, various waste streams, etc. as distinct commodities.

Unfortunately, multi-commodity NFPs are much more general and complex. In an M-commodity problem, we solve for the total flow vector

$$x = (x(1), \ldots, x(M))$$

subject to commodity-wise flow constraints,

$$\sum_{j|(i,j)\in A} x_{ij}(m) - \sum_{j|(j,i)\in A} x_{ji}(m) \; = s_i(m), \forall i \in N, \; m = 1, \ldots, M,$$

and another constraint set, X, "which may encode special restrictions for the various commodities" (Bertsekas, 1998, p. 350). The latter constraints are analogous to the flow bounds (Equation 4.2) in the single-commodity form, but they can be much more general because, in addition to bounding the flows of *individual* commodities on a given arc, they can specify the nature and size of various *compound* flows on that arc by bounding sums of several

commodities. For instance, a limit on the amount of enriched material that can pass between two points could be implemented as a bound on a linear combination of the flows of enriched uranium, fresh fuel, used fuel, etc., but not of yellowcake, unenriched uranium, or separated fission products. Thus, the general form of the multi-commodity problem is given by Equation 4.5:

$$\begin{aligned} &\min_x && f(x) \\ &\text{subject to} && x \in X \\ & && \sum_{j|(i,j)\in A} x_{ij}(m) - \sum_{j|(j,i)\in A} x_{ji}(m) = s_i(m), \\ & && \forall i \in N,\ m = 1, \ldots, M \end{aligned} \tag{4.5}$$

Strictly speaking, this is probably the most appropriate NFP for the network formed by the facilities of the nuclear fuel cycle. But consider the nature of the commodity-wise flows between fuel cycle facilities. Note that if we choose two arbitrary fuel cycle facilities, we know that either no arc will connect them (mines send no material directly to reactors, for instance) or, if an arc does exist, we know precisely which (single) commodity will flow on it (for example, the only material that will travel from a conversion plant to an enrichment plant is unenriched uranium hexafluoride). The only exception is the case of a facility that sends two different types of waste to the repository; I will address later on the unique challenges posed by the repository.

The highly simplifying effects of the observation in the previous paragraph allow us to derive a more practical formulation of the routing problem. We can reduce the very difficult M-commodity problem to M single commodity problems if we first show that both the constraints and the objective function in our application are separable and we then decompose the sets A and N in accordance with that separability to arrive at M separate networks. Of course, we could simply have asserted *a priori* the applicability of Equation 4.4 to the network formed by the buyers and sellers of each fuel cycle commodity and skipped this discussion entirely. But since we have an eye toward global optimization, and since the fuel cycle as a whole is clearly a multi-commodity system, it's worth stepping through these simplifications explicitly so we

can have some confidence that the decision to use LP technology and solve single-commodity problems does not necessarily put us any further from our eventual goal than if we'd used more sophisticated multi-commodity methods.

First of all, note that our fuel cycle system satisfies the conditions of what Bertsekas calls the *separable* multi-commodity problem (1998, p. 350). The constraints are separable because the flow of commodity m on arc (i, j) cannot be constrained by the flow of commodity m' on that same arc if the nature of the facilities associated with i and j and the functioning of the network as a whole are such that no commodity other than m should ever flow on (i, j). Similarly, this behavior is more than sufficient to ensure the applicability of the cost function given in the following formulation of the separable problem:

$$\begin{aligned} \min_{\mathrm{x}} \quad & \sum_{(i,j)\in A} f_{ij}(y_{ij}) \\ \text{subject to} \quad & x_{ij}(m) \in X_{ij}(m), m = 1, \ldots, M \\ & \sum_{j|(i,j)\in A} x_{ij}(m) - \sum_{j|(j,i)\in A} x_{ji}(m) \ = s_i(m), \\ & \forall i \in N, \ m = 1, \ldots, M \\ & y_{ij} = \sum_{m=1}^{M} x_{ij}(m), \ \forall (i,j) \in A \end{aligned} \tag{4.6}$$

Again, if our observation about realistic flows in the fuel cycle is true, than only a single commodity will ever flow on each arc (i, j). Thus, there is only one term in the summation defining each total flow y_{ij}, so we can define $f_{ij}(y_{ij})$ to simply equal $a_{ij}y_{ij}$, where a_{ij} is an appropriate arc cost for the flow of that single, predictable commodity on arc (i, j).

Finally, let N_m and A_m be the commodity-specific node and arc sets. For instance, when m represents mixed-oxide fuel, the nodes in N_m are MOX fuel fabricators and MOX fuel-powered reactors, and the arcs in A_m are the possible links between those fabricators and reactors. We

can use the subsets to specify the terms from Equation 4.6 that drop out under this assumption:

$$x_{ij}(m) = \begin{cases} x_{ij} & \text{if } (i,j) \in A_m \\ 0 & \text{if } (i,j) \notin A_m \end{cases}$$

$$X_{ij}(m) = \begin{cases} [b_{ij}, c_{ij}] & \text{if } (i,j) \in A_m \\ 0 & \text{if } (i,j) \notin A_m \end{cases}$$

$$s_i(m) = \begin{cases} s_i & \text{if } i \in N_m \\ 0 & \text{if } i \notin N_m \end{cases}$$

$$y_{ij} = \begin{cases} a_{ij}x_{ij} & \text{if } (i,j) \in A_m \\ 0 & \text{if } (i,j) \notin A_m \end{cases}$$

With these definitions in place, we can rewrite Equation 4.6 (one separable problem) as Equation 4.7 (M *distinct* problems):

$$\forall m \in \{1, 2, \ldots, M\} :$$

$$\begin{aligned} \text{min} \quad & \sum_{(i,j)\in A_m} a_{ij}x_{ij} \\ \text{subject to} \quad & x_{ij} \in [b_{ij}, c_{ij}],\ \forall (i,j) \in A_m \\ & \sum_{j|(i,j)\in A_m} x_{ij} - \sum_{j|(j,i)\in A_m} x_{ji} = s_i, \forall i \in N_m \end{aligned} \tag{4.7}$$

So far, we've turned the nuclear fuel cycle into M separate networks. Except at the extreme ends of the fuel cycle, each facility serves as a source node in each network corresponding to that facility's `inventory` materials and a sink node in each network corresponding to its `stocks` materials. But even with this notion (and Equation 4.7) in hand, we don't have a complete MRP solution strategy. We've handled the part of the routing problem caused by the discrete *facilities*, but we've yet to handle the fact that the flows themselves must be discrete because we model *materials* discretely as well. For most facilities, we cannot simply turn monthly capacities into static and continuous supply and demand data for setting divergences

because actual supply and demand will depend on the month-to-month status of various facilities' needs—needs that will be met by discrete shipments of material during the time of that need. In other words, we need to choose a *time horizon*, a period over which the networks we construct will be truly representative of the dynamic status of the facilities that comprise the various commodity-wise node sets.

Consistent with the pattern of identifying simple strategies first and working toward more complex ones, I've implemented a naive formulation with the shortest-possible time horizon; *each month*, the manager (with the help of the solver wrapper) constructs and solves an LP formulation of the network problem for each of the commodities for which there is currently supply or demand. Offers of materials or services get turned into sources; the quantity given in the offer message becomes the (positive) divergence value of the node the supplier facility represents. Similarly, requests for materials or services get turned into sinks; the quantity given in the request message becomes the (negative) divergence value of the node the customer facility represents[3]. The solution to the NFP then represents a set of matches of each customer request to a specific supplier's offer, and the manager issues instructions for the suppliers to execute orders according to the nonzero mass flows, x_{ij}, in that solution. A discrete material object representing the matched order then gets constructed appropriately by the supplier and shipped to the customer.

4.3 Materials routing problem: Discussion and modeling details

This approach represents a credible if not completely rigorous first-step toward global optimization of GENIUSv2 material flows. Although the objective function does not calculate system-wide electricity costs explicitly, we can think of this formulation as minimizing the cost of providing all reactors with the materials they need to operate at capacity[4]. When we

[3] Strictly speaking then, we note that while source nodes represent an entire facility making an offer based on its full capacity, a customer facility can be represented by several sink nodes, since each request gets turned into a node and a customer can file multiple requests in the same month.

[4] This description would be even more accurate if the uranium mines and conversion plants operated only in response to orders that had resulted, directly or indirectly, from reactor fuel orders. That's already how enrichment and fabrication plants, which currently do no "extraneous" work, operate. We currently allow mines and

do not wish to penalize the overproduction of electricity or simulate scenarios where material costs are so high that it is advantageous to allow reactors to sit idle, minimizing the cost of keeping the reactor fleets as well fueled as possible is a reasonable approximation to minimizing electricity costs explicitly. For cases where these two considerations *are* important, we may still be able to choose arc costs that capture the desired effects. Otherwise, we might need to develop an entirely new (and probably much more sophisticated) MRP approach.

There are two additional drawbacks to the approach outlined above. First, this approach is naive—it carries the assumption that whenever demand exists, it should be satisfied at the *current* time if at all possible. Because offers and requests arrive on a month-to-month basis, clearing them on that same timescale is the most straightforward option. However, while we almost certainly do not want reactors to sit idle for extended periods of time, there may be cases where the manager would do well to allow a short delay in order to secure a cheaper upstream supply chain. This approach is not sophisticated enough to identify those situations. We can attempt to address this problem going forward by carefully shifting offer and request times earlier and establishing a system for lumping many months' worth of capacity together, allowing us to solve the flow problems for longer time horizons. Of course, the instructions inherent in the network solutions would then need to be appropriately "amortized" over that same number of months.

Second, the decision to continually reconstruct, solve, and discard the mathematical representation of the various NFPs is potentially inefficient, especially the current one-month time horizon. An interesting area for future work is to examine which networks change very little from solution to solution and somehow take advantage of that stability to eliminate redundancy in problem setup and iteration to solution. For instance, we could begin by checking the optimality of the previous solution each time we need to solve a new problem that includes the

---

conversion plants to operate at capacity in the absence of downstream demand so they can build up reserves of material, since most of the scenarios we're interested in modeling involve reactor fleet expansion that eventually results in high material demand.

same nodes as in that previous problem. Similarly, according to the CLP User Guide, the library can be easily extended in order to create dynamic matrix instances for applications where the problem structure evolves over time (2004).

Thus, we simply note that we can't claim to have solved the MRP optimally without either (1) an objective function that explicitly calculates and minimizes the system cost of electricity or (2) a derivation that shows how some form of the flow-cost-minimizing approach can guarantee minimum electricity production costs. However, for the time being we can take full advantage of the reasonable system currently in place to work toward other important aspects of the modeling work GENIUS is intended for.

4.3.1 Affinity-based arc costs and interaction rules

We have so far discussed the *form* of our network flow models of the fuel cycle but only some of the problem *data* for those NFPs. The source and sink nodes and their divergences are determined from the offers and requests that the manager receives at each time step. But how do we determine the flow bounds and arc costs? Well, the obvious physical analog to flow bounds in our system are the constraints (technical and/or regulatory) imposed by the various rail and highway transportation networks that connect different facilities. Although we have not explicitly implemented flow-bound constraints at this time (except to say that the flows must be nonnegative) doing so is now nearly trivial from a coding perspective and merely requires modelers to choose appropriate bounding values.

More interesting for the time being is the choice of arc costs. Because the form of Equation 4.7 seeks to minimize the total cost of the flow of each commodity, the time-step-wise MRP solution can be very sensitive to our choice of these costs. We envision that future versions of GENIUSv2 will include sophisticated economics modules that can choose arc costs via some appropriate combination of (1) the R-I-F "identity" of the nodes they connect, (2) tabulated data representing real-world or user-provided costs (or cost distributions) for the various commodities, such as are available in the *Advanced Fuel Cycle Cost Basis* report (Shropshire, 2007), (3) long-term contracts and other real-world cooperation mechanisms that can be modeled more

or less directly, and (4) accounting-based approaches that set costs dynamically according to the financial and cash-flow situation of the supplier[5].

In describing the arc-cost system currently in place, let me again emphasize that the strength of GENIUSv2 is not that it currently accomplishes anything very sophisticated but that its data structures and encapsulated design sets it up to be a useful test bed going forward. As the manager and solver wrapper construct the NFP used for determining each commodity's routing, they need only make a single function call to determine the cost of each arc. What goes on in that function call can eventually be as complex as we need, perhaps comprising some aspect of each of items (1)-(4) above. For now, the call returns what we term an *affinity*.

The affinity-based costing mechanism is our simplified way of allowing the user to specify complex sets of behavior for material trade within the R-I-F hierarchy. A high affinity for trade between two facilities results in a low cost on the arc connecting them, and vice versa. For now, the affinity scale ranges from zero to ten, with special behavior corresponding to the extreme ends of the scale. If two facilities have a trade affinity of zero for a given commodity, the manager will not even construct an arc between them when setting up the routing problem; if their affinity is ten, then the supplier will automatically be instructed to fill the customer's request before the network is even built.

Affinities are determined by a collection of *rules*. Specified in the input file, a rule manually sets the affinity for trade of a given commodity (or collection of commodities) between a supplier region, institution, or facility and a customer region, institution, or facility. If desired, a start and end date for the rule can also be specified. The manager stores the interaction rules for the simulation in its `RuleBook`, which enforces a consistent precedence convention and will return the affinity between any two facilities subject to the set of rules that currently applies. If no rule does, the `RuleBook` returns a default affinity based on the R-I-F set-theoretical identity of the would-be customer and supplier. These default affinities are given in Table 4.1.

[5]This final approach most resembles the basic mass-flow/cash-flow model proposed by Jain and Wilson (2006, Fig. 1).

Table 4.1 Default trade affinities for determining arc costs.

Customer-supplier relationship	**Affinity**
Both are facilities owned by the same institution	8.0
Facilities are owned by different institutions in same region	5.0
Facilities are in different regions	2.0

Even this relatively simple system provides the means for richly describing patterns of fuel cycle system behavior because the idea of affinities is meaningful but flexible. For instance, we may make the modeling decision that an international corporation can be represented by a series of distinct institutions in each region with a rule that says the affinities for trade between each of them are equivalent to if they were all a single institution. Similarly, we might represent a long-term trade agreement between a fuel cycle region and a customer region via a time-varying mutual affinity that is set appropriately high throughout the duration of the agreement. Much of the content of the following results chapter involves demonstrating the GENIUSv2 R-I-F matching capabilities.

4.3.2 Feasibility and fungibility

With appropriate arc costs in place, two final concerns remain for modeling the network for some commodity m. The first is that supply of commodity m may outstrip demand, or vice versa. As I mentioned above, such is very likely to be the case for each commodity at each time step in realistic problems, so we must take measures to eliminate this fundamental source of infeasibility. Fortunately, a standard approach exists; we add an artificial source or sink to each network that provides the necessary supply or demand to guarantee that the divergences sum to zero. Flow on the artificial arcs (to which we assign appropriately high costs to prevent their being used except when necessary) are ignored when the manager and solver wrapper translate the NFP solution into a set of instructions; however, because these artificial flows effectively measure unmet demand or unused supply, these values will likely be a useful metric for assessing a given fuel cycle design. The addition of these artificial arcs and

nodes completes our schematic picture of the network GENIUS constructs at each time step for each commodity. This schematic is shown in Figure 4.3.

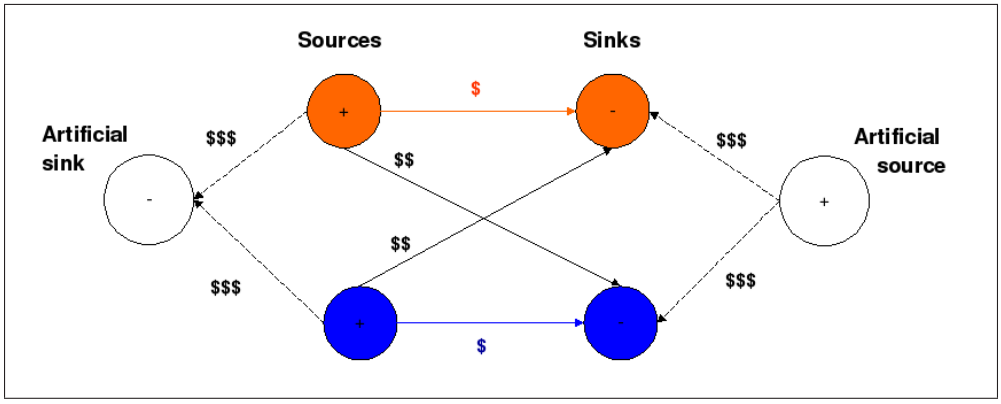

Figure 4.3 Schematic of single-commodity network for GENIUSv2 material routing. Inexpensive arcs connect facilities with a higher affinity for trade. Expensive arcs connect real facilities to artificial ones and are added to guarantee problem feasibility. Image adapted from Oliver et al. (2009).

We deal with flows of infungible material like enriched uranium by letting suppliers offer services and customers order material. Thus, an enricher makes an offer based on its available capacity (in tons-SWU) and a fabricator makes a request based on the exact enrichment and total mass it requires to construct a given fuel recipe. The manager then converts that request into a matchable quantity in tons-SWU by calculating how much enrichment capacity would be required to produce the requested material. Now the source and sink divergences have the same units and can be matched in the given way. Similarly, a separations plant makes an offer of how much total mass of some recycled fuel material it will be able to provide (separated mixed oxides, say). The fuel fabricator orders that commodity according to an exact recipe, which the manager converts to a total mass and matches on that basis (see Section 4.4 below).

If, as in the case of fuel orders, we do not want requests for infungible materials to be spread across two different suppliers, we can manually check the NFP solution and refile any orders that were split, printing a warning when this less-than-ideal situation occurs. We discuss a few more satisfying approaches to this formulation problem in Chapter 6's outline of important future work.

4.3.3 Flows to the repository

It's perhaps appropriate that the final lingering challenge we address is the problem of nuclear waste and how to get it to repositories. As I mentioned above, the repository class is unique in that each non-repository in the fuel cycle may (in theory) wish to send material there. In fact, each may (again, in theory) wish to send multiple commodities there. Thus, the arcs that connect those facilities to repositories appear to violate our separability assumptions, as does the notion that the repository (modeled as a sink node with a single negative divergence) can accommodate multiple commodities. We can easily define this problem away for once-through scenarios with mass-limited repositories; simply treat all waste forms as a single commodity (call it `waste`), and let the repository request a mass of waste each month that does not exceed its remaining capacity. We can generalize this scheme to length-based repositories using a mass-to-length conversion similar to Radel's (2007).

Closed fuel cycles present a bigger challenge; we can't homogenize waste as above if two different facilities (repositories and separations plants) are interested in some types of waste (used fuel) but not all types. Moreover, it's unclear how we make the "competition" between these two "customer" facilities fair and meaningful. Obviously, the case of the repository strains our model a bit, and so we are forced again to assign a satisfactory treatment to the future work queue. For now, the best we can do is name used fuel as a distinct commodity and have separations plants request it and repositories request all other types of waste, including any un-recyclable wastes that emerge from separations. In other words, we must temporarily assume closed fuel cycles in which all waste that *can be* recycled *is* recycled. Careful and creative thinking should yield a mechanism for commodifying and measuring waste in ways that support modeling of the repositories via the "real-estate" assumption that forces waste producers to compete for repository space and thus incentivizes reprocessing (see Radel (2007), Grady (2008)).

4.4 Recipe approximation problem

We transition now to another reprocessing-related modeling challenge, the so-called "winery" issue that I renamed the recipe approximation problem (RAP) when I introduced it in Section 2.2.1. Unlike the materials routing problem, the RAP is not unique to DF/DM codes, but our discrete paradigm does raise some additional questions about how to solve it. These were introduced in Section 3.1.4.2 and are related to the issues of fungibility that arise when modeling continuous processes with discrete materials.

Whatever size is eventually deemed appropriate for the storage of the various streams of reprocessed material at a separations plant, we will call each of these collections a "barrel." For our purposes in solving the RAP, the only necessary assumptions about them are that the material within a barrel is homogeneous and that, while a fraction of the material in a barrel can be removed for inclusion in a material order, the fractional isotopic composition of that removed material must be fixed. In other words, no matter how you slice it, the relative proportions of each isotope remains constant for each chunk or measure of the barrel's material that gets removed[6]. Any satisfactory treatment of the winery issue must eventually satisfy this assumption—otherwise we model highly suspect chemical or physical extraction of reprocessed stream constituents.

We begin to develop a form for solving the approximation problem as follows: Let B be the number of barrels from which the plant may choose in approximating a recipe. At the time of the order, let M_{bi} be the mass of isotope i in barrel b. If we thought we could construct the recipe exactly, we would need only choose the fraction, x_b of each barrel to use such that the aggregate material has mass r_i of each isotope i in the recipe. Determining x in that case would simply require solving the matrix equation

$$Mx = r \tag{4.8}$$

[6]To put it still another way, it's as if the material in the barrel were a chemical compound; naturally, the stoichiometry of the whole is the same as that of any sample you remove.

However, there is no *a priori* reason to suspect that a unique solution exists, nor indeed that any at all do. For instance, we know that no exact solution exists if there are any "impurities" in all our barrels, that is, any isotopes i for which $r_i = 0$ but $M_{bi} \neq 0,\ \forall b \in 1, \ldots, B$. Instead, we allow an approximate recipe $Mx \neq r$ and we define the residual of the approximation:

$$Mx - r \tag{4.9}$$

The smaller we can make the residual, the better the approximation of the original recipe. Ferris, Mangasarian, and Wright (2008, p. 221-222) note that we can cast this problem of minimizing the L^1 norm of the residual ($\|Mx - r\|_1$) as the following linear program[7] by adding dummy variables, y:

$$\begin{aligned} \min_{x,y} \quad & e^T y \\ \text{subject to} \quad & y = |Mx - r| \\ & 0 \leq x_b \leq 1,\ \forall b \end{aligned} \tag{4.10}$$

where e is a vector of ones with length B. The value of the objective function can be used to evaluate the quality of the approximation with respect to the original recipe, and we can set a lower bound on its value to prevent separations from delivering completely unsuitable material.

We can make several improvements to this selection scheme, some more practical than others. The most obvious would be to use the L^2 norm of the residual for the objective function. This choice would give us the common "least-squares" approximation, which has the advantage of more heavily penalizing larger deviations from the given recipe. It would be fairly straightforward to do so using a statistics library; to solve the revised problem with optimization software would require a library capable of solving quadratic programs.

However, one advantage of the LP approach in this context is that it allows us to weight the contribution of each isotope to the objective function and to impose additional constraints. These capabilities are useful in light of three obvious criticisms of Equation 4.10:

[7] In the unlikely case where an exact recipe exists, the objective function value will achieve its absolute minimum of zero, so our approximation assumption does not preclude finding an exact solution to 4.8.

1. There may be orders-of-magnitude differences in the masses of various isotopes called for in the recipe and of isotopes not called for in the recipe but present in some or all of the barrels. The form of the objective function in Equation 4.10 gives an isotopic "matching incentive" that is in direct proportion to the specified mass of that isotope, which isn't really the weighting we desire. For instance, U-238 is almost always the most abundant isotope in a recipe by more than a factor of ten. But neutronic importance is not dependent on mass alone, and we actually care much more about matching the (smaller) U-235 mass than the U-238 mass[8].

2. While the formulation gives explicit incentive to match the mass of each individual isotope, there is not necessarily any additional incentive for the *total* mass of the recipe and of its approximation to match. We want to be careful not to alter the total mass of the core significantly by providing fuel batch approximations whose total mass varies greatly from the recipe's.

3. The formulation does not account for the total neutronic effects of deviation from the recipe. This is the most difficult aspect of the winery problem for any code and obviously the most important one; any recipe approximation is worthless if a core made of the resulting material wouldn't perform properly in a reactor.

The first issue is the easiest to address; we simply normalize the contribution of each isotope to the objective function via a coefficient, c_i formed from the inverse of the mass specified in the recipe (see Equation 4.12). This penalizes mass deviation from the recipe in more appropriate proportions and gives the algorithm more initial incentive to correctly match, say, U-235 (a highly reactive species which in most recipes comprises some small percentage of the total mass) as U-238 (the significantly less reactive majority component in most recipes). For isotopes that appear in the candidate barrels but *not* in the recipe, r_i equals zero and so we cannot divide by it. For now, our coefficient for weighting the penalty for these isotopes is

[8] At least for thermal reactor fuel recipes.

$1/m_r$, where m_r is the total mass of the recipe. Thus, we now have

$$\begin{aligned} &\min_{x,y} && c^T y \\ &\text{subject to} && y = |Mx - r| \\ & && 0 \leq x_b \leq 1,\ \forall b \end{aligned} \tag{4.11}$$

where

$$c_i = \begin{cases} 1/r_i & \text{if } r_i \neq 0 \\ 1/m_r & \text{if } r_i = 0 \end{cases} \tag{4.12}$$

Other choices are possible and may be more appropriate; the present choice merely allows us to minimize our dependence on coefficients that do not have a straightforward physical interpretation.

Next, we add additional constraints to check that the approximation is suitable from the perspective of both total mass and neutronic performance. A strict conservation of mass constraint would be

$$mx = m_r$$

where m_r is the total mass of the recipe, m is a row vector, and the component m_b is the mass of barrel b. However, as we begin to add more constraints, we must worry about our inadvertently making the LP infeasible[9]. Thus, we soften the constraint to

$$|mx - m_r| \leq \epsilon_m \tag{4.13}$$

where ϵ_m is some mass by which it is tolerable for the approximation's total mass to deviate from the recipe's. Within the code, we can set it as some set fraction (say, 2%) of the recipe mass; we can even make that fraction an input parameter. The larger we set ϵ_m (and ϵ_w below), the greater our chances of preserving a non-null subspace of $\mathbb{R}^B$ from which to choose an optimal x.

[9]Duality theory seemed beyond the scope of this treatment of linear programming. The more precise term would be *primal infeasible*

We can attempt to preserve the overall neutronic behavior of the approximation in the same way. Thus, we write another constraint

$$|wx - w_r| \leq \epsilon_w \tag{4.14}$$

where w and w_r are some neutronics-based weighting value (discussed below) calculated for each barrel and for the recipe, respectively. Again, ϵ_w expresses some tolerable deviation in the sum of the weights for the chosen approximation's components from the weight of the recipe. This gives us a final formulation of

$$\begin{aligned} \min_{x,y} \quad & c^T y \\ \text{subject to} \quad & y = |Mx - r| \\ & 0 \leq x_b \leq 1, \ \forall b \\ & |mx - m_r| \leq \epsilon_m \\ & |wx - w_r| \leq \epsilon_w \end{aligned} \tag{4.15}$$

The latter constraint is problematic. It makes perfect sense to enforce conservation of mass via Equation 4.13 because mass is an extrinsic property. However, neutronic properties like cross-sections are intrinsic properties whose values don't change as you vary the size of the sample, so it's unclear how this weighted sum approach will work. We can observe this problem formally by examining the definition of the neutron reproduction factor, η, which is the weight we currently use for w_r and for each w_b. This choice, which of course comes from the four-factor formula for calculating the multiplication factor of an infinite and homogeneous core, is an obvious and appealing place to start because it is the primary measure of the fuel's contribution to the neutron economy and depends *only* on the composition of the fuel—not the core geometry or the composition of the moderator or coolant.

Unfortunately, η doesn't fit very well into our formulation. To see why, first note that for an I-isotope material, the neutron reproduction factor can be expressed according to the following

equations:

$$\eta = \frac{\sum_{i=1}^{I} \nu^i \sigma_f^i n^i}{\sum_{i=1}^{I} \sigma_a^i n^i} = \frac{\sum_{i=1}^{I} \nu^i \sigma_f^i N^i/V}{\sum_{i=1}^{I} \sigma_a^i N^i/V} = \frac{\sum_{i=1}^{I} \nu^i \sigma_f^i N^i}{\sum_{i=1}^{I} \sigma_a^i N^i} \tag{4.16}$$

where σ_f^i, σ_a^i, ν^i, and n^i are the microscopic fission cross-section, the microscopic absorption cross-section, the number of neutrons created per fission, and the number density of isotope i. To make the form suitable for GENIUS, which stores numbers of atoms rather than number densities, we can substitute $n^i = N^i/V$, where N^i is the number of atoms of isotope i and V is the material volume, which we don't need to know because it cancels. Next, let's expand Equation 4.14 from its matrix form into a more explicit summation form:

$$|\sum_{b=1}^{B} w_b x_b - w_r| \leq \epsilon_w \tag{4.17}$$

Ideally, if we could choose barrel fractions x_b such that the composition of the approximation perfectly matched the recipe, then we would want the left-hand side of Equation 4.17 to equal zero. Equivalently,

$$\sum_{b=1}^{B} w_b x_b = w_r \tag{4.18}$$

However, if we let $w = \eta$, define $N^{i,b}$ as the number of atoms of isotope i in barrel b, and substitute Equation 4.16 into Equation 4.18, we can see that this is not quite the case:

$$\sum_{b=1}^{B} \left(\frac{\sum_{i=1}^{I} \nu^i \sigma_f^i N^{i,b}}{\sum_{i=1}^{I} \sigma_a^i N^{i,b}} \right) \Bigg|_b x_b \neq \frac{\sum_{i=1}^{I} \nu^i \sigma_f^i \sum_{b=1}^{B} N^{i,b} x_b}{\sum_{i=1}^{I} \sigma_a^i \sum_{b=1}^{B} N^{i,b} x_b} \tag{4.19}$$

In plain English, the neutron reproduction factor of the whole does not equal the weighted sum of the neutron reproduction factors of the parts we build it with. Again, we shouldn't be surprised by this; there's no reason to expect the reproduction factor to behave like an extrinsic quantity if the cross-sections that comprise its definition aren't extrinsic.

As a concrete illustration of how these constraints work together with the objective function (and, in particular, of the adverse effects of the approximation of Equation 4.19), consider the

following simple case. We wish to construct a recipe containing 100 tons of uranium enriched to 4.5 w/o U-235 using two 100-ton barrels: one enriched to 3.5 w/o U-235 and one enriched to 5.5 w/o. The target recipe has a reproduction factor of $\eta_r = 1.23648$, and the reproduction factors of the candidate barrels are $\eta_1 = 1.13485$ and $\eta_2 = 1.31540$. By inspection, we know that the correct answer to this problem is to use half of each barrel. However, if we fully enforce both the neutronics and the total mass constraint, the solution we desire is not in the feasible region because

$$(0.5)(1.13485) + (0.5)(1.31540) = 1.22512 \neq 1.23648$$

In a sense, the formulation believes that the actual correct solution (which achieves the absolute minimum of the objective function value, zero, because each isotope matches exactly) has a reproduction factor that is too low, because the left-hand side of Equation 4.19 underpredicts the correct reproduction factor (which is given by the right-hand side of Equation 4.19). The algorithm must work from the feasible set of choices for $\{x_1, x_2\}$, which includes only those choices for which the total masses match and for which it believes (incorrectly) that the reproduction factors match.

Table 4.2 shows what happens as we relax and then entirely disregard the neutronics constraint. As the allowable deviation between the calculated η_{approx} and the given η_r gets larger, the feasible region expands until it eventually includes the solution $\{x_1 = 0.5, x_2 = 0.5\}$. The table shows that for this problem, that occurs when ϵ_w/η_r is somewhere between 0.5% and 1%. The first two times times the neutronics constraint is relaxed, the solver can include more U-238 and less U-235, improving the objective function while staying within the limits of acceptable deviation from what it believes to be the desired η_{approx}. The pattern of improvement of course ceases once the feasible set includes $\{x_1 = 0.5, x_2 = 0.5\}$, since the objective function cannot improve after that point.

Conversely, Table 4.3 shows what happens when we instead relax the total mass constraint. In this case, we never expect to recover the correct answer because that answer remains infeasible as long as $\epsilon_w = 0$. But as we begin to allow approximations whose total mass deviates somewhat from the total mass of the recipe, we again see that the solver identifies solutions

Table 4.2 Effects of relaxing the neutronics constraint.

Total mass constraint?	**Neutronics constraint?**	**Calculated solution**	**Calculated** $\eta_{\mathbf{approx}}$	**Actual** $\eta_{\mathbf{approx}}$	**Enrichment [w/o U-235]**
Enforced: $\epsilon_m = 0$	Enforced: $\epsilon_w = 0$	$x_1 = 0.437103$ $x_2 = 0.562897$	1.23648	1.24748	0.0462579
Enforced: $\epsilon_m = 0$	Enforced: $\epsilon_w = 0.005\eta_r$	$x_1 = 0.471344$ $x_2 = 0.528656$	1.23030	1.24154	0.0455731
Enforced: $\epsilon_m = 0$	Enforced: $\epsilon_w = 0.01\eta_r$	$x_1 = 0.5$ $x_2 = 0.5$	1.22512	1.23648	0.045
Enforced: $\epsilon_m = 0$	Not enforced	$x_1 = 0.5$ $x_2 = 0.5$	1.22512	1.23648	0.045

that allow it to include less U-235 and more U-238, steadily improving the objective function. Unfortunately, as the total mass constraint gets very loose, the objective function continues to improve even as the approximation passes the desired enrichment. Thus, we come very close to the correct enrichment at $\epsilon_w/\eta_r = 1\%$ but then continue into an underenriched regime as the tolerance moves on toward 2%.

Table 4.3 Effects of relaxing the total mass constraint.

Total mass constraint?	**Neutronics constraint?**	**Calculated solution**	**Calculated** $\eta_{\mathbf{approx}}$	**Actual** $\eta_{\mathbf{approx}}$	**Enrichment [w/o U-235]**
Enforced: $\epsilon_m = 0$	Enforced: $\epsilon_w = 0$	$x_1 = 0.437103$ $x_2 = 0.562897$	1.23648	1.24748	0.0462579
Enforced: $\epsilon_m = .005m_r$	Enforced: $\epsilon_w = 0$	$x_1 = 0.47353$ $x_2 = 0.53147$	1.23648	1.24157	0.0455765
Enforced: $\epsilon_m = .01m_r$	Enforced: $\epsilon_w = 0$	$x_1 = 0.509957$ $x_2 = 0.500043$	1.23648	1.23561	0.0449018
Enforced: $\epsilon_m = .02m_r$	Enforced: $\epsilon_w = 0$	$x_1 = 0.538137$ $x_2 = 0.475731$	1.23648	1.23097	0.0443845
Not enforced	Enforced: $\epsilon_w = 0$	$x_1 = 0.538137$ $x_2 = 0.475731$	1.23648	1.23097	0.0443845

This "overshooting" occurs because the normalization strategy for choosing objective function coefficients, c_i, favors matching the U-235 exactly ($c_{235} = 1/4.5$) over matching U-238 exactly ($c_{238} = 1/95.5$). Although in general we do care more about matching the mass of U-235 than the mass of U-238, in a two-isotope problem like this one it's really this single *ratio*

of masses that we want to get right. However, the formulation does not encode this preference, and so it takes advantage of the slack afforded by loosening the total mass constraint and endeavors to match U-235 exactly at the expense of the (less valuable) U-238 match. Figure 4.4 plots this behavior, decomposing the objective function value into the component contributions from the U-235 and U-238 terms over a range of tolerances. Notice that the best enrichment achieved (that is, the one that's closest to the enrichment of the recipe) occurs when the contributions of each term are closest to equal.

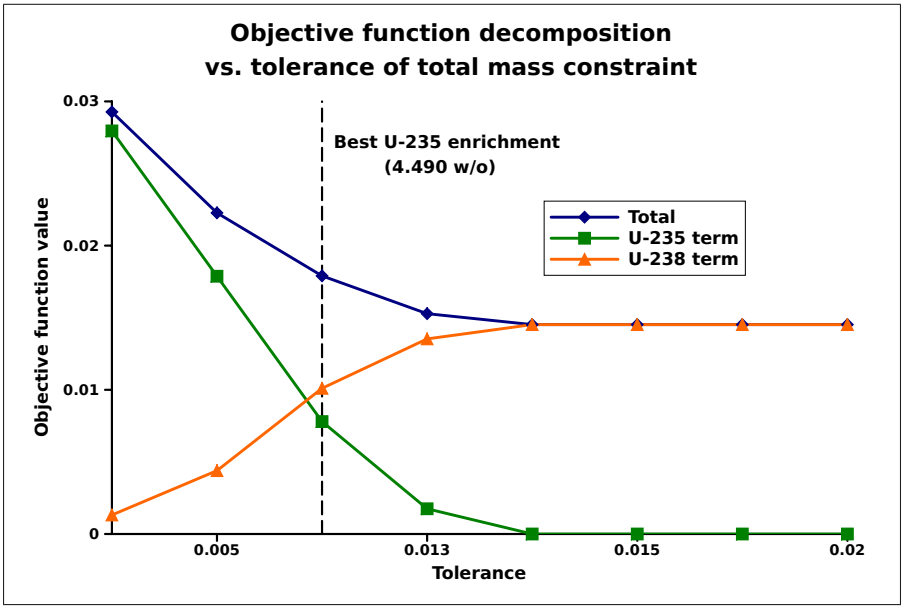

Figure 4.4 Decomposition of the objective function value into its U-235 and U-238 contributions. Note that the algorithm favors U-235 matching because that isotope's objective function coefficient is greater.

Although in the example above the approximations would be best served by removing the neutronics constraint altogether, it's not unreasonable to expect that using this η-based neutronic-weight constraint will improve the quality of some recipe approximations, since it still carries first-order information about which isotopes are favorable from a reactivity perspective and which aren't. We shall see in the results chapter that Equation 4.15 does do a serviceable job in many cases. Again, this formulation can be thought of in a sense as placeholder functionality that allows us to begin to model closed fuel cycles *without* modeling non-physical

extraction from a single homogenized collection of reprocessed material. The approach can be refined and made more rigorous from a reactor-physics perspective via subsequent research projects, or it can be replaced entirely if necessary.

4.5 Summary

This chapter derived a series of single-commodity linear network flow programs to solve the multi-commodity routing problem on a month-to-month basis. Drawbacks of and possible future directions for this formulation were briefly discussed, the most important drawback being the naive time horizon. The chapter also presented an approximation-theory based linear program for solving the recipe approximation problem that governs the process of constructing recycled fuel from reprocessed material. Here the most significant drawback is the less-than-ideal form for the neutron weighting constraint. Chapter 5 will test and demonstrate the results of these formulations and better inform future attempts to improve them.

Chapter 5

Test problems and demonstration results

This chapter reports results from various testing and benchmarking problems designed to demonstrate the capabilities in place in GENIUSv2. More rigorous and systematic testing and the first realistic fuel cycle calculation for a real-world client are ongoing. The first major tests run on GENIUS were (1) benchmark problems designed to check its total mass flow results against VISION's for a set of comparable reactor deployment scenarios and (2) a pair of large multi-region scenarios designed to illustrate the R-I-F hierarchical matching methodology described in Section 4.3.1. In this chapter, Sections 5.1.1 and 5.1.2 report on these testing and demonstration activities and are drawn, mostly verbatim, from the conference paper[1] that reported on them (Oliver et al., 2009). The remaining sections give new results not yet reported elsewhere.

5.1 Once-through fuel cycle results

5.1.1 Comparisons with VISION

Because VISION has emerged as the standard tool for performing fuel cycle systems analysis calculations, the following benchmarking problems compare GENIUS and VISION results for a series of analogous test problems of increasing complexity. Throughout the discussion it will be important to remember that *analogous* is the operative word. At a certain level of

[1]Many thanks to my coauthors on that paper for their suggestions and feedback, and in particular to CNERG's VISION expert Tae Wook Ahn, who ran the problems on the VISION side in addition to helping me identify appropriate fuel cycle parameters (see Table 5.1) and the form for Equations 5.1 and 5.2. Thanks also to Katy Huff for her help with GENIUSv2 recipe management.

granularity, fleet-based, continuous-flow codes and DF/DM codes are simply incompatible. However, the overall behavior, including integrated material throughputs, of each model's approximation of any given scenario should give nearly to the same answers. Note that each problem was run with VISION's radioactive decay routines turned off, since the analogous routines in GENIUS are still being tested and debugged.

Problem 1 is a single-reactor benchmark with no fuel fabrication constraints[2], and its purpose is to compare accumulated spent fuel mass and isotopics in GENIUS and VISION. The parameters for this simulation and the three that follow it are given in Table 5.1 and represent a combination of parameters that in our experience tend to work well in the VISION model.

Table 5.1 Parameters for VISION-GENIUS benchmark problems.

Parameter	**Value**
Start year	2000
End year	2099
Construction + license time	6 years
Operating time, OT	60 years
Power capacity, P	1050 MWe
Capacity factor, CF	0.90
Thermal efficiency, η	0.34
Cycle time, T	12 months
Fuel burnup, Bu	51 GWd/tHM
Fuel batches per core, N	5

Comparing GENIUS and VISION results requires first devising a method to ensure that their reactors' fresh and spent fuel isotopics are comparable. To accomplish this task, one must reconcile the different means by which each code accounts for material. VISION stocks and flows are each described by a total mass whose isotopic breakdown is stored as a set of mass fractions, with each fraction corresponding to a particular isotope. The discrete materials in GENIUS, on the other hand, store the *absolute* number of atoms of each isotope, not the atom or mass fraction. Thus, to construct a pair of GENIUS fuel recipes, first determine the fixed

[2]GENIUSv2 provides mechanisms for creating special unconstrained testing facilities that make offers according to specified monthly capacities and execute the corresponding orders immediately upon matching. The materials themselves get created out of thin air, so to speak. If one of these test facilities has its capacity to some extremely large number, there is no risk of insufficient fuel being available.

core mass, M, corresponding to the material in a VISION reactor fleet representing a single reactor. Next, multiply M by the VISION mass fraction vectors that represent fresh and spent fuel isotopics for a particular fuel type and burnup level. Finally, convert the corresponding masses into numbers of atoms and load the resulting GENIUS-compatible recipes into the code. The only difficulty in this procedure is dealing with isotopes that VISION does not track individually. These VISION recipe constituents, which get tagged with the label _OTHER, cannot be converted unambiguously into GENIUS recipe constituents; while they have a well defined mass in VISION (their mass fraction times the total mass of the stock or flow), they cannot have such in GENIUS because in GENIUS the mass of a constituent is stored implicitly as the number of atoms times an appropriate atomic mass. While some effort was made to assign representative atomic masses to the _OTHER components, the process introduced some error (see below).

The GENIUS fresh and spent fuel isotopics for these problems are based on the mass fractions from a standard VISION LWR fuel recipe through interpolation for 51 $\frac{GWd}{tHM}$ of burnup. To obtain M, first note that VISION calculates a continuous fuel consumption rate by quarter-year time step according to Equation 5.1:

$$\dot{m} = \frac{P(CF)}{\eta(Bu)} \tag{5.1}$$

This amount of mass, which constitutes $\frac{1}{N}th$ of the total core, emerges from the reactor during the cycle period, T, so the total core mass for a GENIUSv2 reactor using this recipe is

$$M = \dot{m}NT = \frac{PNT(CF)}{\eta(Bu)} = 99.459\ \frac{tHM}{core} \tag{5.2}$$

In Problem 1, reactor construction and licensing start in 2000, and once the reactor begins operating it runs for its designated operating time before being decommissioned. We can do a simple hand calculation to compute the total mass ejected from a reactor for an idealized real-world refueling scheme: At startup, N batches are inserted (total mass M). During each normal year of operation one batch is inserted and one ejected (batch mass $\frac{M}{N}$). In the decommissioning year, all N batches currently in the core are ejected (total mass M). Thus, the total ejected

mass, M_{ej}, should be (neglecting mc^2 losses):

$$M_{ej} = \frac{M}{N}(OT - 1) + M = 1.273\ ktHM \tag{5.3}$$

Table 5.2 shows the results of the hand calculation and the error in the two simulations with respect to that prediction, and Figure 5.1 shows the codes' behavior as a function of time. The GENIUS underestimate for the total is due to a conservation of mass violation in our procedure for reproducing the appropriate amounts of _OTHER isotopes from the VISION discharge isotopic recipe. The VISION overestimate may result from its particular implementation of modeling a discrete process like refueling in a continuous manner, although note also that an extra offloading of one batch (mass $M/N = 19.9$ tons) would approximately account for the difference. Overall, though, this analysis suggests that VISION is a suitable reference against which to compare GENIUS and indicates the magnitude of discrepancy that can be expected in comparing more complex scenarios.

Table 5.2 Comparison of total spent fuel mass calculations for single reactor.

Calculation method	**Total ejected fuel mass [ktHM]**	**Relative error**
Hand calculation	1.273	–
VISION simulation	1.293	+1.57%
GENIUS simulation	1.267	-0.47%

This reasonable agreement extends to the isotopic level. Figure 5.2 plots results for the discharge isotopics of the five largest actinide streams in both simulations. The end-of-simulation discrepancies are of the same magnitude as when we compare total masses; the differences in the GENIUS results with respect to the VISION results fall between 1.81% and 2.05% (see first column of Table 5.3 below).

To ensure that these results scale appropriately, for Problem 2 we repeated the same test but with ten such reactors. Because these reactors were identical to the first one and all begin operating at the same time as in the single case, we expected and observed magnitudes exactly ten times greater than in the single reactor case and with the same end-of-simulation discrepancies.

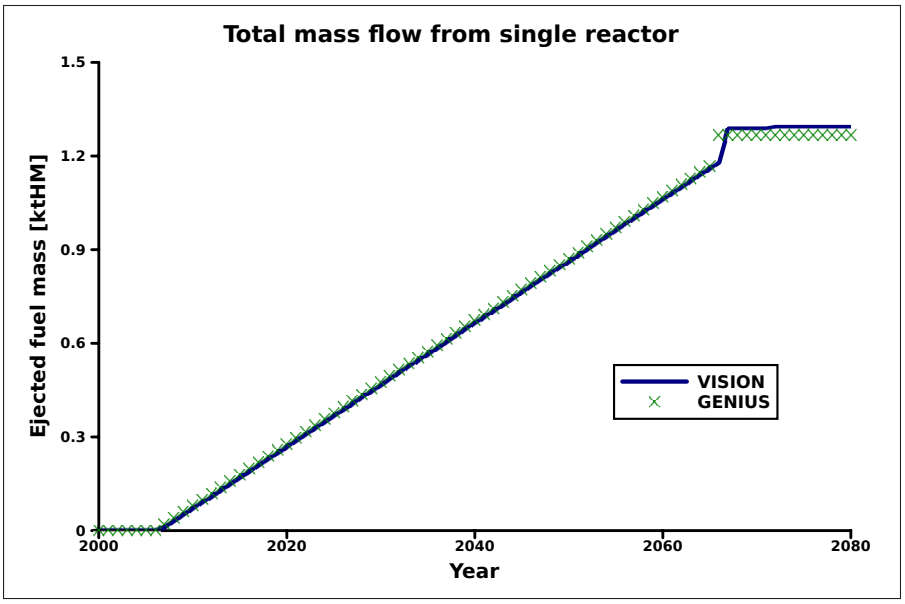

Figure 5.1 Integrated total mass flow from the single reactor in Problem 1. Because GENIUS is a discrete-flow code, it handles reactor startup and decommissioning in a straightforward way. Image and caption from Oliver et al. (2009).

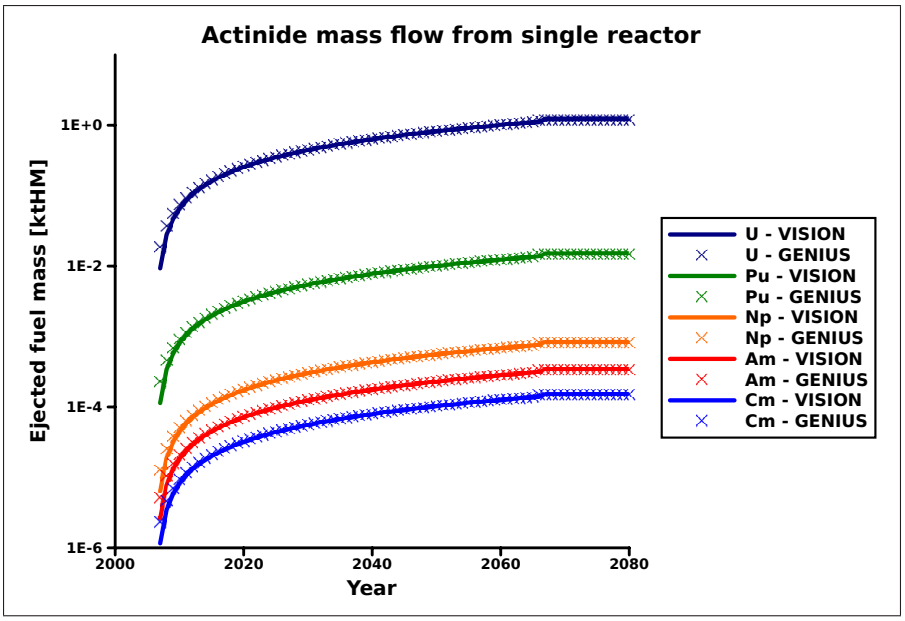

Figure 5.2 Integrated mass flow from the single reactor in Problem 1, for the five largest actinide streams (note semi-log scale). The end-of-simulation errors in the GENIUS results with respect to the VISION results for each element are given in the first column of Table 5.3. Image and caption from Oliver et al. (2009).

Problems 3 and 4 complicate the facility deployment by specifying growth curves. As mentioned earlier, GENIUSv2 requires a user-driven facility deployment in order to avoid using a deployment heuristic. Of course, calculating a reactor deployment to meet an arbitrary demand curve is a fairly trivial problem, so that capability has been written into the GENIUSv2 pre-processor. Thus, Problem 3 is a stepwise-linear growth case where the simulation starts building reactors in 2000 and increases the total capacity by one reactor each year (requiring two new reactors per year starting in 2067 to account for one retirement per year during those final 34 years).

The mass flow results for this problem are shown in Figure 5.3 (with isotopic breakdowns again in Table 5.3), and they once again show good agreement. Notice that that the VISION-GENIUS discrepancy shrinks measurably and that this time GENIUS gives a larger result. Careful examination of Figure 5.1 suggests an explanation. Because of the differences in how they handle startup and decommissioning, the time-integrated ejected mass values for GENIUS reactors are higher than for VISION during most of the reactor's lifetime (because VISION only ejects half a batch's worth of fuel in the first year) but are lower than for VISION once decommissioning is complete (because VISION reactors consume more total lifetime fuel than their GENIUS counterparts, as seen in Table 5.2). Thus, some of the error cancels. And because 94 of the 128 total reactors built during the simulation have not yet been decommissioned in 2100, the VISION fleet lags behind the GENIUS fleet in terms of fuel mass ejected so far, even though in the end each of the VISION reactors will have used slightly more fuel than the GENIUS reactors.

Problem 4, the final VISION-GENIUS benchmark problem, is for exponential growth in electricity demand. The initial demand is 10 GWe and the demand growth rate is 2% per year. The simulation includes ten "legacy" reactors that exist when the simulation starts. They retire, one per year, starting in 2029. The mass flow results for the VISION and GENIUS simulations of this deployment are given in Figure 5.4. Here GENIUS returns a lower total mass than VISION; as noted in Problem 3, the higher the percentage of total reactors that reach decommissioning by the end of the simulation, the more likely VISION is to compute

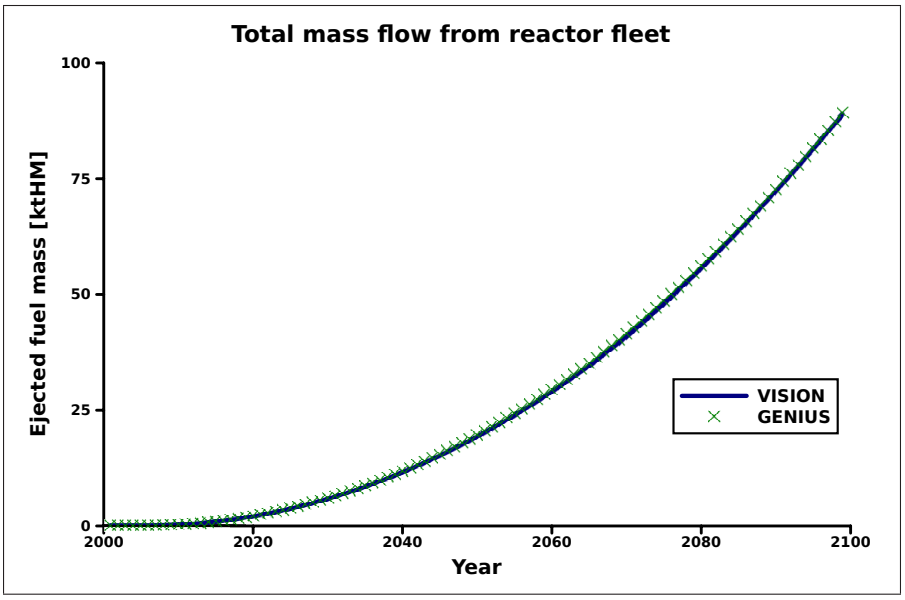

Figure 5.3 Integrated total mass flow from the reactor fleet in Problem 3. The end-of-simulation errors in the GENIUS results with respect to the VISION results for total mass and the five main actinide streams are given in the third column of Table 5.3. Image and caption from Oliver et al. (2009).

larger mass flows than GENIUS. In Problem 4, 31.7% of the reactors get decommissioned, as opposed to only 26.6% in the previous problem, so it's not very surprising that GENIUS returns to computing a lower total mass output than VISION. Finally, note that the isotopic discrepancies with respect to the VISION case are given in the final column of Table 5.3 and once again show reasonable agreement at that level of detail as well.

Table 5.3 Summary of isotopics results for benchmark problems.

	GENIUS error w/r/t VISION			
Material stream	**1**	**2**	**3**	**4**
Total mass	-2.00%	-2.00%	0.32%	-0.53%
Uranium	-1.93%	-1.93%	0.38%	-0.47%
Plutonium	-2.05%	-2.05%	0.27%	-0.58%
Neptunium	-1.89%	-1.89%	0.43%	-0.43%
Americium	-1.91%	-1.91%	0.40%	-0.45%
Curium	-1.81%	-1.81%	0.51%	-0.34%

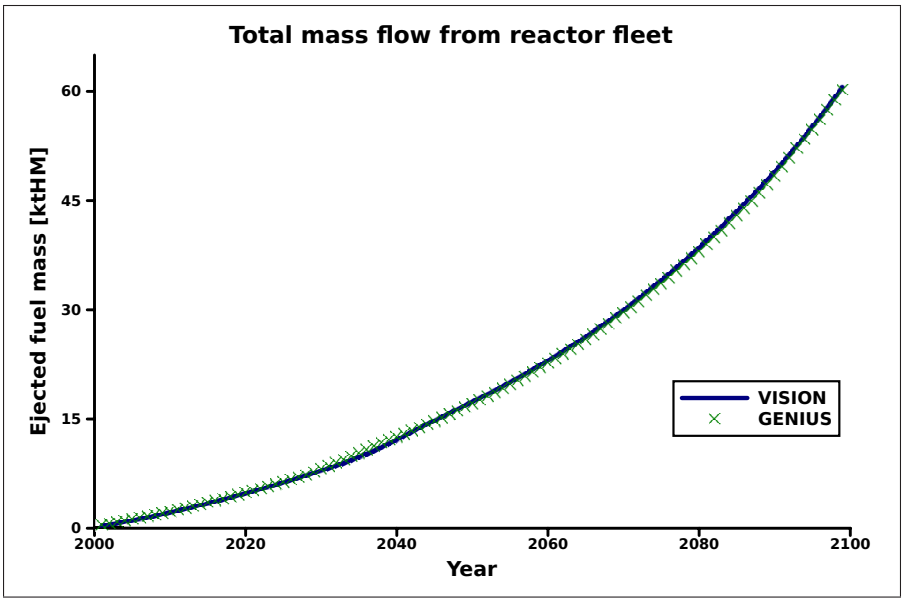

Figure 5.4 Integrated total mass flow from the reactor fleet in Problem 4. The end-of-simulation errors in the GENIUS results with respect to the VISION results for total mass and the five main actinide streams are given in the fourth column of Table 5.3. Image and caption from Oliver et al. (2009).

5.1.2 Rule-based fabrication matching in three-region problem

Problems 5 and 6 are two once-through problems that demonstrate GENIUSv2's ability to be scaled up to larger scenarios and that point to the richness and flexibility of the R-I-F model and the formulations for solving the MRP. The facility deployment for Problems 5 and 6 is given in Table 5.4. Both of the reactor regions contain three institutions: a small and a large fuel fabricator and reactor operator building either PWRs or PHWRs to match a linear demand curve. The third region contains only a large fabricator with facilities for both LWR and PHWR fuel. The parameters for both types of reactors are given in Table 5.5. The precise fuel fabrication capacities were chosen to match reactor batch sizes in order to avoid the unrealistic scenario of splitting a fuel batch order between two different fabricators. Future work will explore optimization techniques to relax this constraint in a way consistent with the network flow model. Note that this is a fairly large problem. It calls for construction of 748 total reactors (compared to 130 in Problem 3, the largest of our VISION benchmark problems) and records 43,521 individual material transfers (9,612 in Problem 3).

Table 5.4 Facility deployment for three-region fuel fabrication matching problem.

Region	Institution	Facilities
1	1	1 PWR in Jan. 1970 Linear growth: 1.71 GWe/year
	2	1 LWR Fuel Fab (78.66 tHM/month)
	3	1 LWR Fuel Fab (157.3 tHM/month)
2	4	1 PHWR in Jan. 1970 Linear growth: 675 MWe/year
	5	1 PHWR Fuel Fab (333.7 tHM/month)
	6	1 PHWR Fuel Fab (667.3 tHM/month)
3	7	1 LWR Fuel Fab (157.3 tHM/month) 1 PHWR Fuel Fab (667.3 tHM/month)

Table 5.5 Parameters for three-region fuel fabrication matching problem.

	Value	
Parameter	**PWR**	**PHWR**
Start year	1970	
End year	2099	
Construction + license time	5 years	
Operating time, OT	50 years	
Capacity factor, CF	0.90	
Power capacity, P [MWe]	1000	600
Thermal efficiency, η	0.33	0.30
Cycle time, T [months]	18	12
Fuel burnup, Bu [GWd/tHM]	45	7
Fuel batches per core, N	4	1

In Problem 5, the GENIUS matching algorithm solves the MRP according to the default affinities described in Section 4.3.1. Problem 6 alters the default behavior by specifying two rules: Institutions 1 and 4 (the reactor operators) get preferentially matched with Institution 7 (the extra-regional fuel fabricator) as if they were all the same institution. This affinity assignment could represent any number of modeling decisions, including to simulate a long-term contract between Institutions 1 and 7 and 4 and 7; to signify that all three are, in fact, owned by the same company; or to capture some price advantage benefiting Institution 7. Figure 5.5 (PWRs) and Figure 5.6 (PHWRs) show results for the cumulative travel of fabricated fuel from the various suppliers to the reactor fleets in the two different problems. The top of

each figure shows the default behavior; the extra-regional fabricator is the supplier-of-last-resort and is only purchased from consistently when the order density is high enough that the fabricators in the reactor regions are always working at capacity. Conversely, the bottom plot of each figure shows that the foreign fabricator is now preferred.

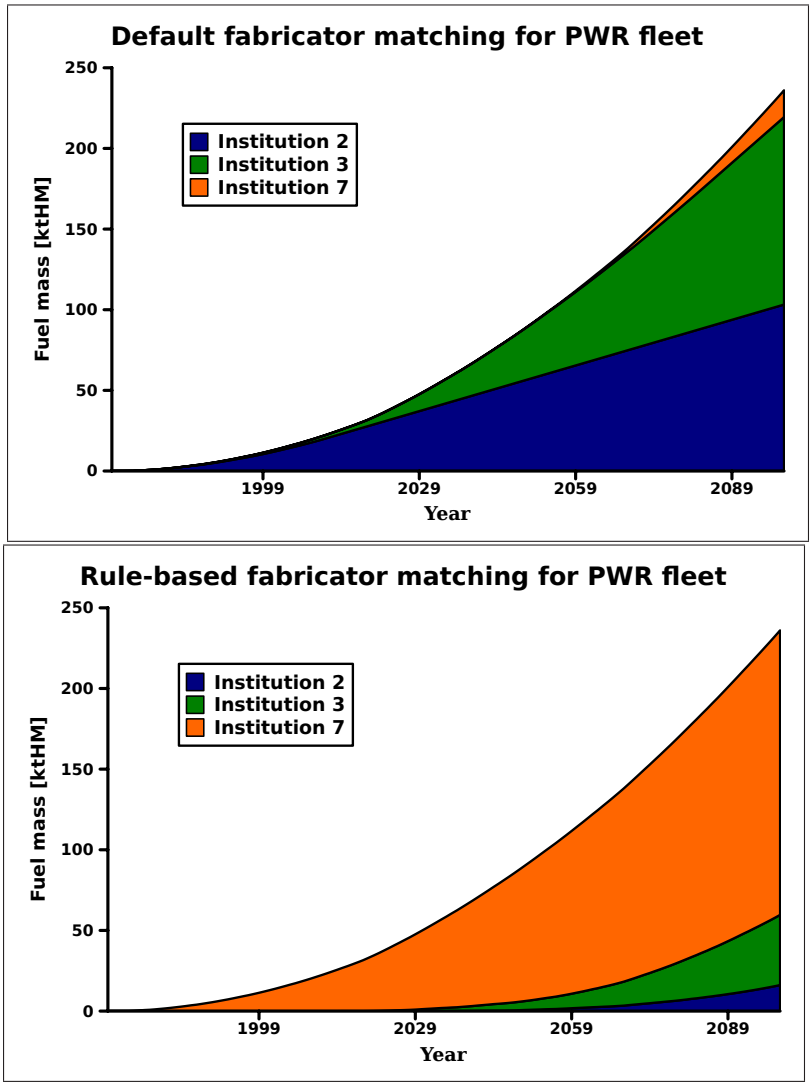

Figure 5.5 Change in matching of PWR fuel fabricators to reactors placing orders. When the reactor operator's affinity for trade with Institution 7 is increased sufficiently, it becomes the favored supplier even though it's located in another region. Image and caption from Oliver et al. (2009).

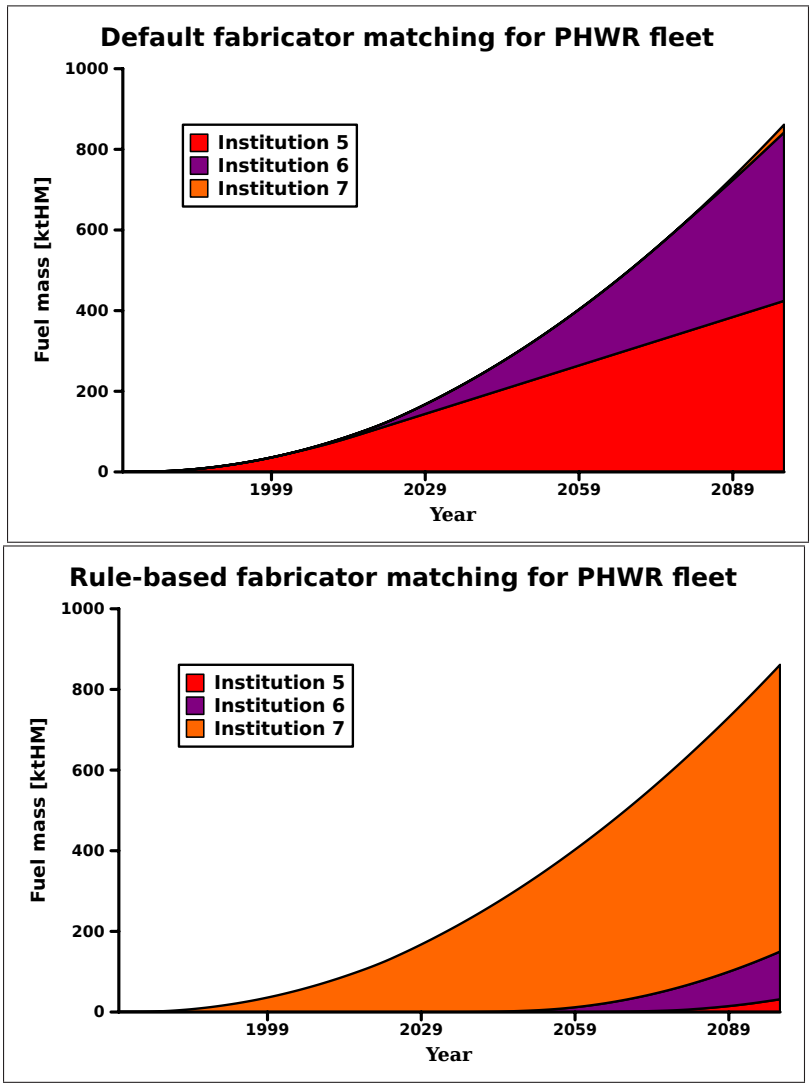

Figure 5.6 Same as Figure 5.5, except for the PHWR region's fleet. Note again the substantially different material routing based on one change to the input file. Image and caption from Oliver et al. (2009).

5.1.3 Rule-based unenriched uranium matching in four-region problem

Problems 7 and 8 present a similar scenario with a somewhat expanded scope to further demonstrate the kinds of modeling and analysis GENIUSv2 is capable of. These problems simulate the entire front end of the fuel cycle with facilities that have large capacities but are constrained by material availability and process times in the usual, realistic ways[3] described in Chapter 3. Thus, we can examine scenarios where the availability—or, more accurately, the source—of raw materials is the object of study.

Table 5.6 shows the facility deployment of this scenario, and the other simulation parameters used are the same as in Problems 5 and 6. There are two modeling decisions of special interest here. First, note that Regions 2 and 3 are home to future reactors but no fuel cycle facilities; these regions are meant to represent a small- and a medium-sized fuel "user state" dependent for their fuel on a "supplier state," Region 1 (recall Figure 2.1). One serious question these user states might ask is how the global uranium market will affect the prices they will have to pay their supplier for fabricated fuel. In fact, this is an interesting scenario even if the users have negotiated some guaranteed price, because then increased material costs would have to be absorbed in the fuel supplier state, possibly being passed on to rate-payers served by the domestic reactor fleet that also depends on Region 1's fuel service providers.

Table 5.6 Facility deployment for four-region unenriched uranium matching problem.

Region	Institution	Facilities
1	1	1 PWR in Jan. 1970 Linear growth: 1.71 GWe/year
	2	1 Mine/Mill, 1 Conversion
	3	1 Enrichment, 1 Fuel Fab
2	4	1 PWR in Dec. 2019 Linear growth: 0.81 GWe/year
3	5	1 PWR in Dec. 2019 Linear growth: 1.71 GWe/year
4	6	1 Mine/Mill, 1 Conversion

[3] In other words, there are none of the special "testing" facilities that were deployed to supply fuel to the reactors in Problems 1–6.

The second notable aspect of this scenario is that it simulates the effects of the kind of price shock that could impact this user/supplier fuel service arrangement. It does so via a mechanism based loosely on the current U.S. situation: First, we model mining and conversion capacity within the fuel supplier region *and* in some foreign region (Region 4). Second, we set up a rule that creates affinities for trade favoring the inexpensive foreign supplier over the pricier domestic supplier, akin to how the U.S. currently chooses to purchase uranium from Russia rather than mining its own[4]. Third, we abruptly change this uranium-trade status quo by perturbing the affinities appropriately. Thus, even though GENIUS does not currently incorporate economic modeling to effect the resulting change in material routing via arc costs drawn from real-world price changes, it can still create approximations to the behavior that would result using its extremely simple affinity-based model.

Problem 7, accomplishes the proper "affinity management" via a single rule: we set the affinity for trade of unenriched uranium hexafluoride from Region 4 to Region 1 at 6.0, effective from the beginning of the simulation until some set expiration date, d (see Table 5.7). Recall from Table 4.1 that the default affinity for inter-institution trade within a single region is 5.0. Thus, early on in the simulation, whenever Institution 3's enrichment plant orders feed material, the manager solves a network flow problem in which the arcs that connects the enrichment plant orders to Institution 2's conversion plant have a higher cost than the ones connecting them to Institution 6's conversion plant. Thus, until this rule expires, the foreign supplier provides all the uranium for Region 1's fuel cycle activities. After it expires, the default inter-region affinity of 2.0 raises the cost on the previously cheaper arc and it is no longer selected. Note that this all-or-nothing behavior takes place because both regions have sufficient capacity to supply fuel to all the reactors in the simulation. If the capacities were lower, we would sometimes have observed the supplier-of-last-resort behavior of Problems 5 and 6.

[4]Of course, Russia sends us uranium downblended from highly enriched weapons stockpiles, not material that has been freshly mined, milled, and converted to UF_6. Although GENIUSv2 enrichment plants could be programmed to downblend when necessary and then model the current arrangement by initializing the foreign enrichment plant with a large stockpile of highly enriched material, it would be a fair amount of work to implement these capabilities for such an unusual and specialized circumstance. Pretending the uranium is freshly mined doesn't affect the nature of the material exchanges between the two regions and is thus an adequate approach for an idealized demonstration problem.

Problem 8 incorporates two more finely tuned rules in order to simulate something like an even market situation when the initial rule expires. There are several ways to accomplish this; the chosen approach is shown in Table 5.7. Here, the rule that expires during the simulation is equivalent to saying that Regions 4 and 1 are the same, at least for the purposes of unenriched uranium trade. But this would set a level playing field right away. To prevent matching of the domestic uranium supplier to the domestic fuel service provider, we *lower* the affinity between Institutions 2 and 3. By lowering it to the default inter-region affinity of 2.0, the arcs connecting the domestic enricher to both the foreign and domestic supplier have the same cost when the first rule expires. The result is "foreign supplier only" behavior until the Rule 1 expiration date, followed by some sharing of orders after that time.

Table 5.7 Summary of matching rules used for Problems 7 and 8.

Problem	**Rule**	**From**	**To**	**Commodity**	**Affinity**	$\mathbf{t_{start}}$	$\mathbf{t_{end}}$
7	1	Region 4	Region 1	uUF6	6.0	0	d
8	1	Region 4	Region 1	uUF6	5.0	0	d
	2	Inst 2	Inst 3	uUF6	2.0	0	1560

Figure 5.7 plots the relevant mass flows in Problem 7 for three different cases representing three different values of d (Rule 1 expires at the beginning of 2010, 2020, and 2040, respectively). The lowermost area represents the cumulative supply from Region 4 in all three cases; the adjacent area above represents the portion that is provided by Region 4 under the two later expiration cases but not the early one (in which it is instead provided domestically); the next layer is the portion provided domestically for both of the early expiration cases but not the latest one; the uppermost area corresponds to the supply provided domestically in each case. No uranium enters Region 1 from Region 4 in Problem 7 after Rule 1 expires.

Figure 5.8 uses the same plotting convention to show the data for Problem 8, subject to the same three different expiration dates for Rule 1. Note in this case that material continues to be transferred between Regions 4 and 1 after time d, but a significant portion of the necessary uranium starts to be supplied domestically at that time. By the end of the simulation, between 32.8% and 38.9% of the total mass has come from the domestic source, with the largest

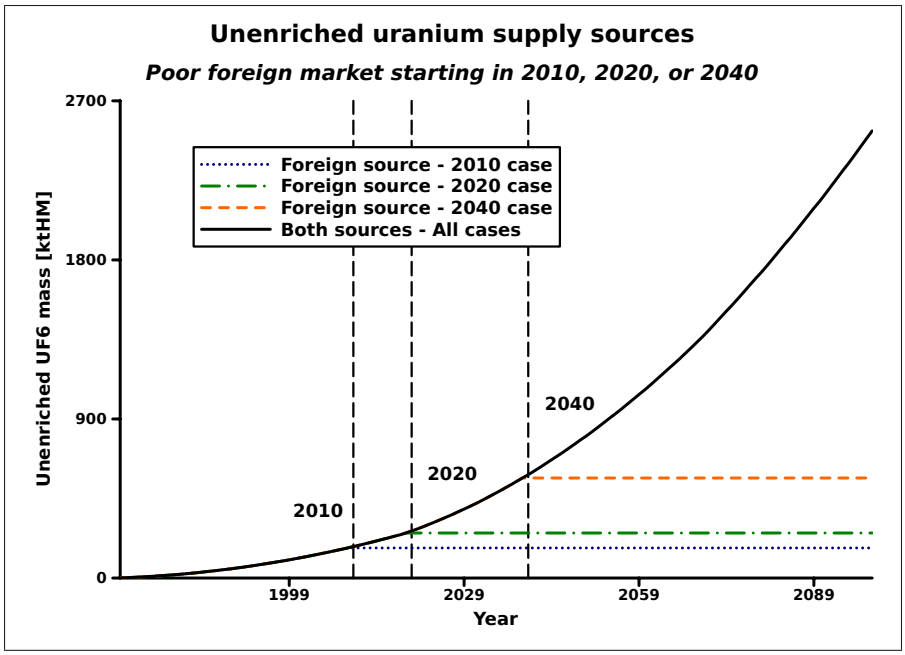

Figure 5.7 Sources of unenriched uranium for domestic enricher and fabricator providing fuel for the PWRs in Problem 7. A rule favors the foreign supplier until its expiration date, at which point the default affinities favor the domestic supplier. This figure plots the three cases where the rule expires in 2010, 2020, and 2040, respectively.

amount obviously corresponding to the earliest expiration date for whatever agreement caused the foreign supplier to be advantageous.

How exactly the orders get distributed after time d is a fair question to ask, and the answer is one that could incline us to change either the network formulation, the way the code passes problems to the solver software, or the simple affinity mechanism, as development continues. From an LP/NFP perspective, the problem is *degeneracy*; multiple optimal solutions can and often do exist. For instance, if each of the R requests can be filled by two different suppliers, and if each supplier has enough capacity to fill all of them, and if all the arcs connecting the suppliers to the requests have the same arc cost—all of which conditions are true after time d in Problem 8—then there are 2^R possible ways to match them. In fact, it seems to be just a happy coincidence that orders get distributed between the two suppliers once the market is even. For instance, the first plots in Figures 5.5 and 5.6 show that in Problems 5 and 6 we were not so lucky. The "even competition" between Institutions 2 and 3 and between Institutions 5

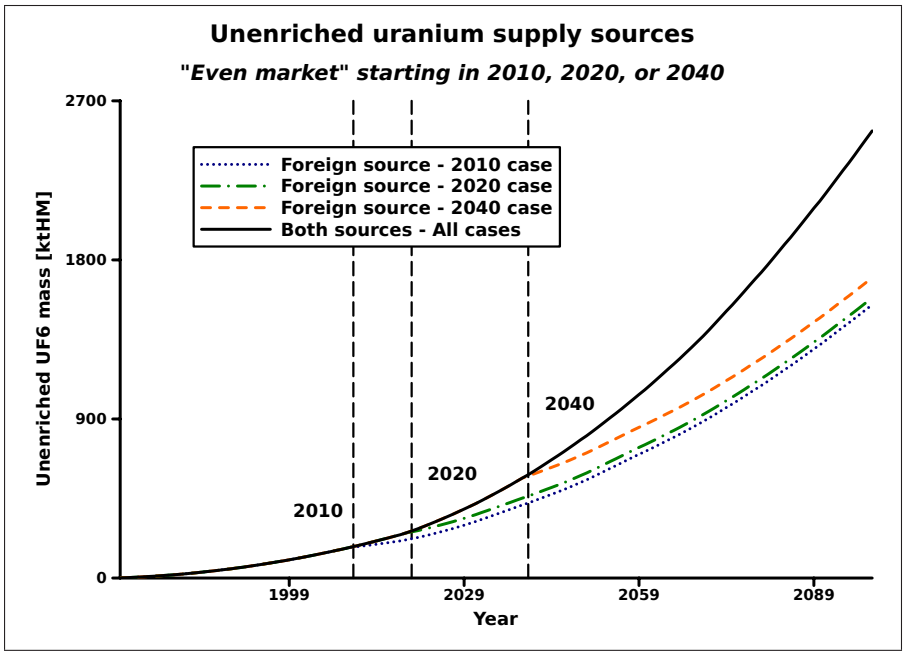

Figure 5.8 Sources of unenriched uranium for domestic enricher and fabricator providing fuel for the PWRs in Problem 8. One rule favors the foreign supplier until its expiration date, at which point only the second rule applies, setting the affinities even. This figure plots the three cases where the first rule expires in 2010, 2020, and 2040, respectively.

and 6 did not result in their sharing orders. Instead, Institutions 2 and 5 were matched with requests as long as they had the capacity and only in months when they were fully committed did Institutions 3 and 6 receive any business, so to speak.

It seems very likely that the systematic way the solver wrapper constructs the mathematical representation of each NFP—combined with the systematic way the solver itself iterates toward a solution—causes the solver to return similar degenerate solutions each time. In the case of Problems 5 and 6, those solutions consistently favored one fabricator over another. The same is somewhat true in Problem 8. For each of the three expiration dates for Rule 1, Figure 5.9 plots the fraction of the sum of the uranium mass provided by the foreign supplier each year (its yearly market share, so to speak). Although the fraction is not constant, it is always greater than or equal to 0.5, suggesting a sort of artificial preference for the foreign supplier due to the

random but seemingly consistent identification of degenerate solutions that draw more uranium from it despite the equal arc costs across the network.

It's interesting but not especially surprising that the ratio does not change from one *case* to the other (note the overlapping data points in Figure 5.9). For a given month m such that $m \geq d$, the NFP solved is almost exactly the same across the three cases—the same number of offers (2) and requests (R) are filed by the same facilities, and the arcs that connect suppliers to clients have the same cost associated with them. Only the quantity of available material that each supplier has to offer (i.e., the divergence of each source node) depends on what has happened in previous months. Of course, the danger in making observations like these is that the solution behavior under degeneracy is likely very algorithm- and library-dependent; one certainly wouldn't want to get in the habit of depending on or trying to predict it.

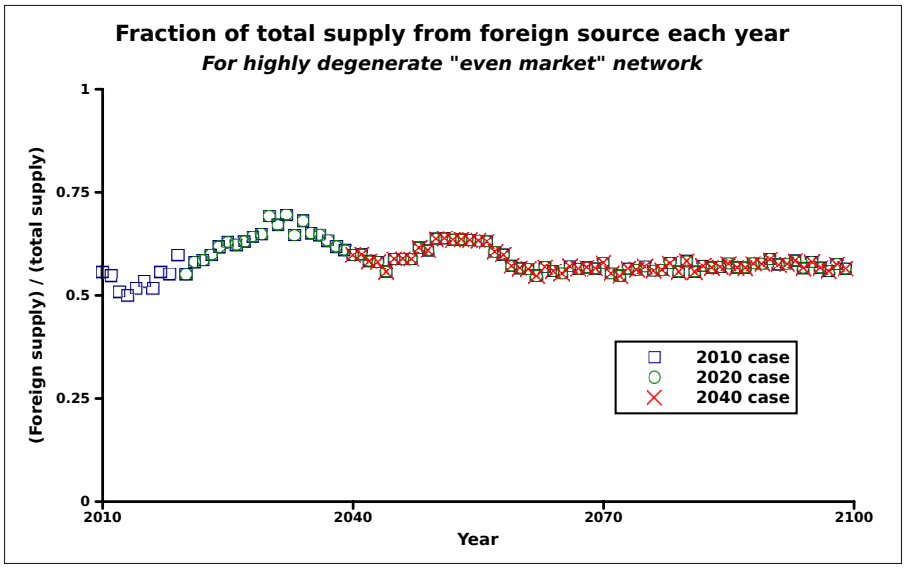

Figure 5.9 Yearly foreign supply over total supply in Problem 8. This fraction is unpredictable because after the Rule 1 expiration date, d, there are 2^R degenerate optimal solutions for the NFP that matches the R uranium requests each month to one of the two suppliers, and we do not know which solution the solver will find.

What to do about degeneracy should depend on careful thought about why (and even if) it truly poses a modeling problem, especially in the long term. We could of course devise methods for removing the degeneracy itself or the effects of the solver's apparent tendency to return

similar degenerate solutions for similar problems. For instance, as long as GENIUS solves the NFP with LP technology anyway, additional constraints could encourage the manager to better balance the load in cases with many equal-cost arcs. We could also probably randomize the order in which we translate the list of offers and requests into the mathematical representation of the problem. Or, if the effects of that measure didn't make it past the solver's pre-processor, we could randomly perturb the cost of each arc by some small but detectable random number.

However, this last strategy hints at a point that may encourage us to simply ignore degeneracy for now: once the code moves beyond a purely affinity-based arc-costing scheme, it's likely that most of the problematic degeneracy will just disappear. For example, if distance between supplier and customer were ever to be made even a small part of the calculation GENIUS performs to assign arc costs, most of those costs would immediately become unique, thus eliminating much of the degeneracy without any artificial or constraint-increasing intervention. It's reasonable to assume that any sophisticated arc-costing schemes will avoid assigning identical costs to very many different arcs.

5.2 Closed fuel cycle results

5.2.1 Recipe approximation unit tests

This section briefly describes the simple unit tests that any form for the recipe approximation problem should certainly pass. Of course, the ability to pass these tests does not guarantee success on more realistic problems, and in fact for more realistic problems it's not often entirely clear what the correct answer is. In a sense, then, the advantage of starting with the tests below is that they serve as a mechanism for screening out approaches that have little hope of being useful (*before* those approaches make it to problems on which it's hard to evaluate their effectiveness). For the tests that follow, the values $\epsilon_w = 0$ and $\epsilon_m = 0.01 m_r$ were chosen due to their relative success in the illustration case from the end of Chapter 4, which is reproduced as Test 6 below.

To test the mass-matching functionality apart from the neutronics considerations that complicate matters so significantly, Tests 1-4 involve only non-fissionable isotopes. Thus, $w_r = 0$

and $w_b = 0, \forall b = 1, \ldots, B$ and so the neutron weight constraint (Equation 4.14) is trivially satisfied. Tests 1-4 are various perturbations on approximating heavy water (D_2O) recipes given candidate barrels of varying usefulness. In Test 1, the algorithm is asked to construct 1 Mmol of D_2O from a barrel containing 1 Mmol of D_2O and another containing 1 Mmol of light water (H_2O). Obviously, we expect the LP solution to tell us to use all of one and none of the other. Test 2 doubles the recipe (2 Mmol D_2O) and adds a third barrel, a duplicate of the match from Test 1; we now want the solution to include both D_2O barrels. Test 3 uses the same candidates but reduces the target recipe by 25% (to 1.5 Mmol D_2O). Of course, this test has a family of degenerate optimal solutions, and we merely require that the solution returned belongs to that family. Finally, Test 4 must approximate Test 1's target recipe using Test 2's candidates. However, to each of the D_2O barrels we've added different amounts of an impurity, "dissolved" helium. The solver should choose the barrel with the least of that impurity. Table 5.8 summarizes Tests 1-4 and establishes the notation used for the remainder of this section.

Table 5.8 Problem definition and results for Tests 1-4: Recipe approximation with non-fissionable components.

Test	**Target recipe [Mmol]**	**Candidate barrels [Mmol]**	**Desired solution**	**Result**
1	[H-1, H-2, O-16] = [0, 2, 1]	1 : [2, 0, 1] 2 : [0, 2, 1]	$x_1 = 0$ $x_2 = 1$	$x_1 = 0$ $x_2 = 1$
2	[H-1, H-2, O-16] = [0, 4, 2]	1 : [2, 0, 1] 2 : [0, 2, 1] 3 : [0, 2, 1]	$x_1 = 0$ $x_2 = 1$ $x_3 = 1$	$x_1 = 0$ $x_2 = 1$ $x_3 = 1$
3	[H-1, H-2, O-16] = [0, 3, 1.5]	1 : [2, 0, 1] 2 : [0, 2, 1] 3 : [0, 2, 1]	$x_1 = 0$ $x_2 + x_3 = 1.5$	$x_1 = 0.0$ $x_2 = 0.5$ $x_3 = 1.0$
4	[H-1, H-2, He-4, O-16] = [0, 2, 0, 1]	1 : [2, 0, .00, 1] 2 : [0, 2, .01, 1] 3 : [0, 2, .02, 1]	$x_1 = 0$ $x_2 = 1$ $x_3 = 0$	$x_1 = 0$ $x_2 = 1$ $x_3 = 0$

While these tests are useful for demonstration purposes and for confirming the basic functionality of future formulations, in practice there will *always* be fissionable isotopes in the recipes and candidate barrels. In Tests 5-7, then, the solver attempts to approximate a simple

uranium oxide fuel recipe weighing 100 tHM and with a U-235 enrichment of 4.5 w/o. Because $\nu^i \sigma_f^i$ is nonzero for the isotopes i involved in these tests, the neutron weighting constraint will affect them.

Test 5 is the loose equivalent of Test 1; the candidate barrels include one containing the correct material and one that is under-enriched. The basic mass-matching constraints should suffice to allow the solver to discern among the two barrels which is correct, but we include this test to be sure that the neutron-weighting formulation does not interfere with this ability. Test 6 replaces Test 5's correct barrel with an over-enriched one with 5.5 w/o U-235. This test was discussed for a number of different parameters at the end of Chapter 4; recall that the obvious correct answer is to use half of each barrel. Table 5.9 shows that the formulation (with $\epsilon_w = 0$ and $\epsilon_m = 0.01 m_r$) does reasonably well on this problem, using roughly 2% more of the under-enriched barrel than it should. Unfortunately, this test points out one of the main drawbacks of our formulation: the nature of the approximation inherent in the neutronic weight constraint is that it often leads to approximations with *smaller* reproduction factors than the target recipe. If anything, the opposite would be preferable, so that the approximation has excess reactivity with respect to the target and thus a better chance of being usable in a real reactor. In the case of Test 6, the error in η is only -0.07%; however, that error increases for Test 7. In this final test, we remove the U-235 from the over-enriched barrel (making it undesirable to use at all) and add a third barrel containing plenty of pure Pu-239. An ideal formulation in this case would instruct us to use all of the barrel enriched to 3.5 w/o U-235 and to add enough of the Pu-239 to recover the original neutron reproduction factor[5]. In truth, the answer approximates that qualitative behavior but again "under-produces" from the perspective of neutron regeneration, this time with an error of -1.8% (a fairly large number given the scale at which deviations from criticality matter). However, unlike in Test 6, where including the neutronic weighting constraint erodes the quality of our approximation, Test 7 would have been worse off without it, since no plutonium at all would be used in its absence.

[5]This amount can be calculated analytically or numerically; it comes to approximately 0.46 tons for this problem.

Table 5.9 Problem definition and results for Tests 5-7: Recipe approximation with fissionable components.

Test	Target recipe [tons]	Candidate barrels [tons]	Desired solution	Result
5	[U-235, U-238] = [4.5, 95.5]	1 : [3.5, 96.5] 2 : [4.5, 95.5]	$x_1 = 0$ $x_2 = 1$ ($\eta = 1.23648$)	$x_1 = 0$ $x_2 = 1$ ($\eta = 1.23648$)
6	[U-235, U-238] = [4.5, 95.5]	1 : [3.5, 96.5] 2 : [5.5, 94.5]	$x_1 = 0.5$ $x_2 = 0.5$ ($\eta = 1.23648$)	$x_1 = 0.509957$ $x_2 = 0.500043$ ($\eta = 1.23561$)
7	[U-235, U-238, Pu-239] = [4.5, 95.5, 0]	1 : [3.5, 96.5, 0.0] 2 : [0.0, 94.5, 0.0] 3 : [0.0, 0.0, 5.50]	$x_1 = 1$ $x_2 = 0$ $x_2 = 0.0836327$ ($\eta = 1.23648$)	$x_1 = 0.989637$ $x_2 = 0$ $x_2 = 0.0615514$ ($\eta = 1.21332$)

Finally, we mention briefly two "pathological" tests upon which this method fails completely because of the intrinsic/extrinsic problem discussed in Section 4.4. Consider first what would happen if we produced a test that had the same relationship to Test 5 as Test 2 did to Test 1. If we double the target recipe and add a barrel identical to Test 5's correct answer, the method will fail; while the correct answer is to use both of the 100-ton barrels enriched to the proper weight percentage of U-235, this choice leads to violation of the neutronic weight constraint because the sum of the candidates' neutron regeneration factors is twice the value of that same factor for the target recipe. Thus, the solver will tell us that the problem is infeasible. Conversely, we could provide several under-enriched (or even just natural) uranium barrels. In this case, even though each provides insufficient neutron reproduction, from the standpoint of our current neutron weight constraint, the sum of these barrels would have an (artificially) adequate combined η value. We can choose to help the separations plants avoid these situations by ensuring that they combine or divide barrels as necessary to maintain candidates that are appropriately sized for the sizes of order the plants are likely to receive. However, this strategy could interfere with any modeling decisions about what the real-world size of a barrel should be. In the absence of this technique, problems of this nature currently require relaxing or removing the neutronic weight constraint.

The above shortcomings emphasize the point that more work is needed in identifying improved formulations for solving the general recipe approximation problem. However, this methodology still represents a substantial improvement over non-physical methods that allow extraction of arbitrary isotopes from a large soup of recycled material. This claim can be strengthened when GENIUSv2 incorporates a simplified burnup engine like that of Scopatz and Schneider (2009), since such a tool would allow deviation from input recipes without introducing such egregious errors in allowable burnup and discharge isotopics when large deviations do occur. In sum, the above method is flawed but nevertheless preferable to the currently available alternatives, at least from a long-term modeling perspective.

5.2.2 A simple recycling scenario

Finally, we turn our attention to studying recipe approximation *in situ*, via a scenario (Problem 9) that includes reprocessing. This activity is meant both to demonstrate GENIUSv2's ability to simulate closed fuel cycles and as an opportunity to further explore and refine the RAP-formulations. Problem 9 models a high-demand and high-growth region served by enriched uranium oxide PWR reactors and a lower-demand, lower-growth region served by mixed oxide PWRs. For the latter, we choose a VISION MOX fuel recipe that contain uranium, neptunium, and plutonium only and is suitable for thermal recycle scenarios. The facility deployment and simulation parameters for this case are given in Tables 5.10 and 5.11, respectively. The fuel cycle facilities all have more than adequate monthly capacity to meet the needs of the reactor fleets. To maximize the illustrative power of this example, we run it without decay, without any lag between spent-fuel ejection and reprocessing, and without allowing the separations plant to discriminate between spent UOX fuel and spent MOX fuel when procuring material to reprocess.

Since Institution 8's fuel fabrication plant produces MOX fuel, it sends requests for a commodity enumerated internally as `sepMox` (for separated mixed oxides). Separations plants currently estimate their availability of this commodity when making offers for it by summing over the material objects that were produced from appropriate streams of separated material. Which

Table 5.10 Facility deployment simple thermal recycle scenario.

Region	Institution	Facilities
1	1	12 UOX PWRs in Jan. 2010 Linear growth: 850 MWe/year
	2	1 UOX Fuel Fab
	3	1 Separations
	4	1 Mine/Mill
	5	1 Conversion
	6	1 Enrichment
2	7	3 MOX PWRs in Jan. 2010 Linear growth: 142 MWe/year
	8	1 MOX Fuel Fab

Table 5.11 Parameters for simple thermal recycle scenario.

	Value	
Parameter	**UOX PWR**	**MOX PWR**
Start year	2010	
End year	2109	
Decay	Turned off	
Fuel cooling delay	None	
Separation plant requests	All used fuel	
Construction + license time	5 years	
Operating time, OT	50 years	
Capacity factor, CF	0.90	
Power capacity, P [MWe]	1050	1050
Thermal efficiency, η	0.34	0.34
Cycle time, T [months]	12	12
Fuel burnup, Bu [GWd/tHM]	51	46
Fuel batches per core, N	5	5

of those streams is appropriate will depend on what kinds of MOX fuels it expects to be providing material for. Because for this scenario we know the separations plant will be providing the material for U-Np-Pu MOX fuels only, we manually set the plant to use an implementation of the VISION 2.2 base case's UREX3 method[6]. This method separates the recycled fuel into (among others) a uranium stream, a neptunium-plutonium stream, and a higher actinide stream, the first two of which are the relevant ones for producing `sepMox` suitable for U-Np-Pu fuels.

[6]Thanks to CNERG's Royal Elmore for his help implementing the separations schemes in GENIUSv2.

Just as we manually set the separations plant to use a process that's appropriate for eventually producing U-Np-Pu MOX, so too do we ensure that it only includes suitable barrels (from the uranium and neptunium-plutonium streams) when setting up a recipe approximation problem to solve. These two choices are a subset of a larger problem that can be labelled "separations tuning," which is itself a subset of the overall "fuel-cycle tuning" issue introduced in Chapter 3. Helping the separations plant figure out how to operate in a sensible way given the recycled material it will likely be called upon to produce is a particularly important area of future GENIUS modeling work and stands to significantly improve the quality of the recipe approximations without having to change the formulation; if candidate barrels have been separated according to a scheme that is tailored to specific recipe of interest, they will be identified and used accordingly.

The scenario outlined above was repeated parametrically for many different combinations of the tolerances ϵ_w and ϵ_m from the RAP formulation given in Equation 4.15. During the simulation, the 27 MOX-fueled PWRs order a total of 929 batches of fuel. The quality of each of these approximations is plotted (in simulation order) in Figure 5.10 and summarized in Table 5.12 for the subset of the (ϵ_w, ϵ_m) parameter space that best illustrates the trend toward reasonably successful approximations. In this series, we set ϵ_m constant at 10% of the total recipe mass and let ϵ_w vary from 10% to 50% of the recipe's reproduction factor before relaxing it completely. This scenario comes the closest to providing recipes with the appropriate reproduction factor when the influence of the η-based neutron weight constraint is completely removed. Given the underestimates of η in Test 6 and 7 caused by the inexactness of Equation 4.19, this pattern is not especially surprising.

On the other hand, the *variability* in Figure 5.10 is at first a little surprising, since in the absence of decay all fuel arriving at separations has one of only two possible recipes. However, if we think about the chief cause of this variability, we can understand both why it exists and why it diminishes over time. The dominant causes here are

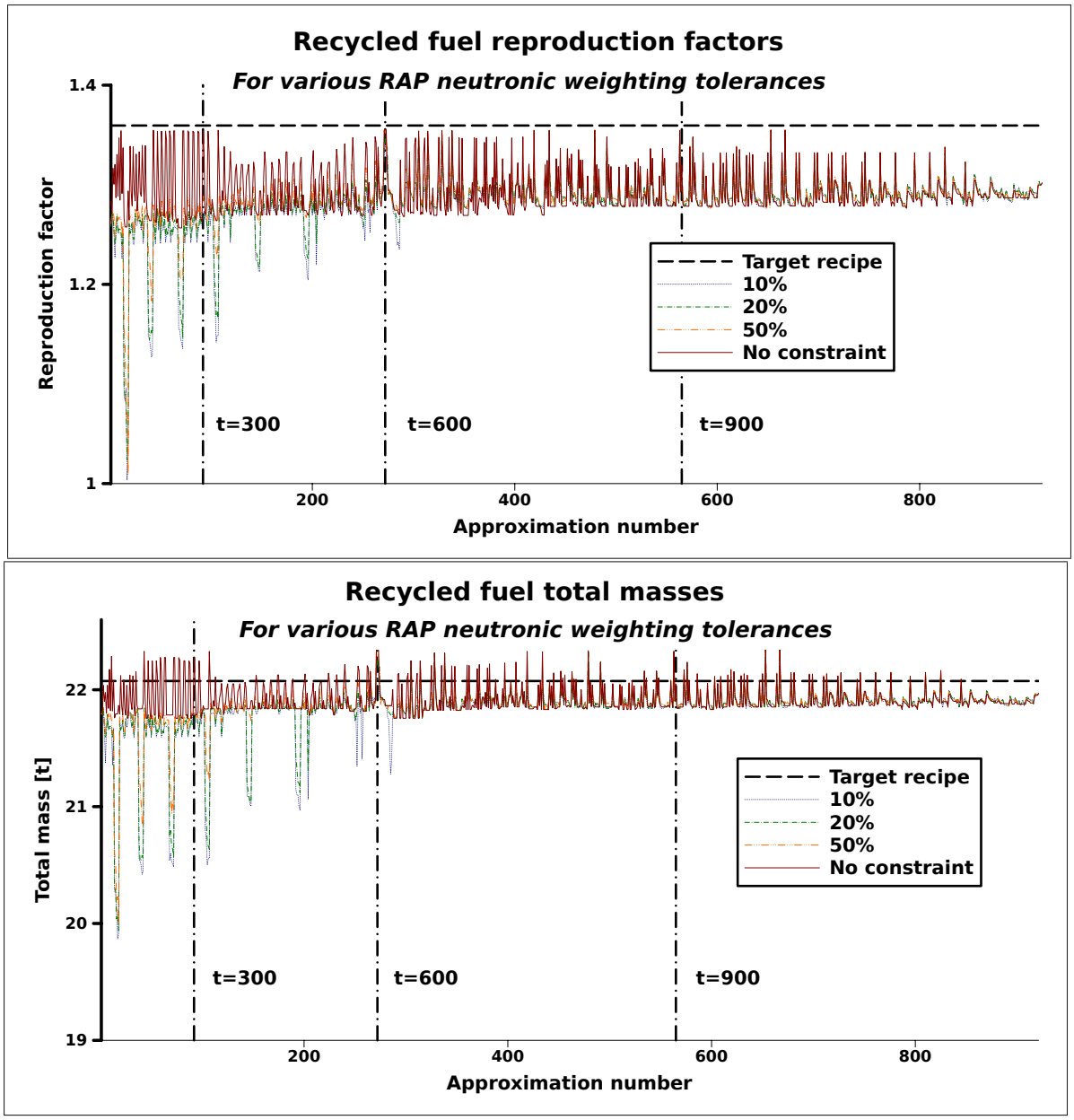

Figure 5.10 Quality of recipe approximations in simple recycling scenario. Unlike in Test 7, which required a neutronics-weight constraint in order to get close to the correct answer, here including this constraint degrades the quality of the approximation.

- the high variability in the number of shipments both to and from separations when there are few reactors in the system (which means the number and size of the candidate barrels changes a lot from month to month at early times),

Table 5.12 Statistics on deviation from desired η and total mass for recipe approximations in simple recycling scenario for various RAP neutron weighting tolerances.

	$\eta_{\mathbf{rec}} - \eta_{\mathbf{approx}}$			$\mathbf{m}_{\mathbf{rec}} - \mathbf{m}_{\mathbf{approx}}$		
Tolerance	**Avg** [%]	**Min** [%]	**Max** [%]	**Avg** [%]	**Min** [%]	**Max** [%]
10%	5.77	0.335	26.1	1.12	-1.25	10.0
20%	5.57	0.333	24.7	1.05	-1.26	9.67
50%	5.19	0.334	25.8	0.892	-1.26	9.42
No constraint	**4.89**	**0.332**	**7.56**	**0.707**	**-1.27**	**1.44**

- the "tuning" between the dominant discharge recipe and the recipe for fresh recycled fuel, and
- the polluting effects of the "twice-burned" spent MOX fuel and the increasing probability that separations will receive some of it in a given month in addition to spent UOX fuel.

The latter two items in the above list are best understood via a side-by-side listing of the composition of desired versus available material. Table 5.13 shows the mass fractions of each isotope that we would like to have available from the barrels formed from the uranium and neptunium-plutonium streams (Column 3) as well as the mass fractions we actually get from (separately) reprocessing spent UOX fuel (Column 4) and spent MOX fuel (Column 5). Notice that the relative proportions within the two streams are closer to matching the desired values for the spent UOX stream than for the spent MOX stream. Moreover, the neptunium-plutonium stream from the spent UOX is a better fit than the uranium from spent UOX. These observations about the quality of the different recycled streams seem to be true from a neutronics perspective as well. The most important isotopes with respect to reactivity are U-235, Pu-239, and Pu-241, so we expect that the approximations with the best neutronic properties will be from when barrels unpolluted by reprocessed MOX fuel are available.

Figure 5.11 helps us see the big picture with respect to the flow of recycled material in this final scenario. By the end of the simulation, almost 5 ktHM of spent fuel have arrived at reprocessing. Less than half of this amount ends up being sent back to fuel fabrication to be burned in MOX reactors. The time axis on the out-going plot has been reversed for ease of

Table 5.13 Composition of desired and available recycled material for constructing MOX fuel.

Stream	Isotope	Compositions [w/o]		
		Desired for fresh MOX	**Available from spent UOX**	**Available from spent MOX**
Uranium	U-232	0	4.27e-9	7.06e-8
	U-233	0	2.14e-8	1.23e-6
	U-234	2.00e-2	4.35e-4	3.16e-2
	U-235	**0.822**	**0.756**	**0.517**
	U-236	0.613	0.599	0.608
	U-238	98.5	98.6	98.8
Neptunium-Plutonium	Np-237	5.03	5.25	4.17
	Pu-238	2.50	2.45	5.79
	Pu-239	**50.4**	**47.0**	**39.3**
	Pu-240	23.9	23.8	27.2
	Pu-241	**11.2**	**14.1**	**14.1**
	Pu-242	6.99	7.37	9.48
	Pu-244	0	2.45e-4	2.10e-4

comparison of the relative proportions of selected isotopes. The key observations here are as follows: First, the relative proportions *within* each of the two streams remain fairly constant and are the same entering the separations plant as leaving it. This observation reiterates the point that the separations plant does not allow *isotopic* separation, only elemental separation. It also serves to remind us that, no matter how good the formulation for recipe approximation is, it can only be as successful as the quality of the candidate material it has to work with. Second, the relative proportions *between* the two streams are not constant; cumulatively, the approximation algorithm chooses to use proportionally less of the uranium stream than the plutonium stream compared to what it has available. This is unsurprising in light of Table 5.13, and it's reassuring from a reactivity perspective because we expect the plutonium stream to be closer than the uranium stream to having the specified neutronic properties.

5.3 Summary

This chapter reported results from a number of testing and demonstration problems for both once-through and closed fuel cycles. Problems 1-4 showed that reactor fleets in GENIUSv2

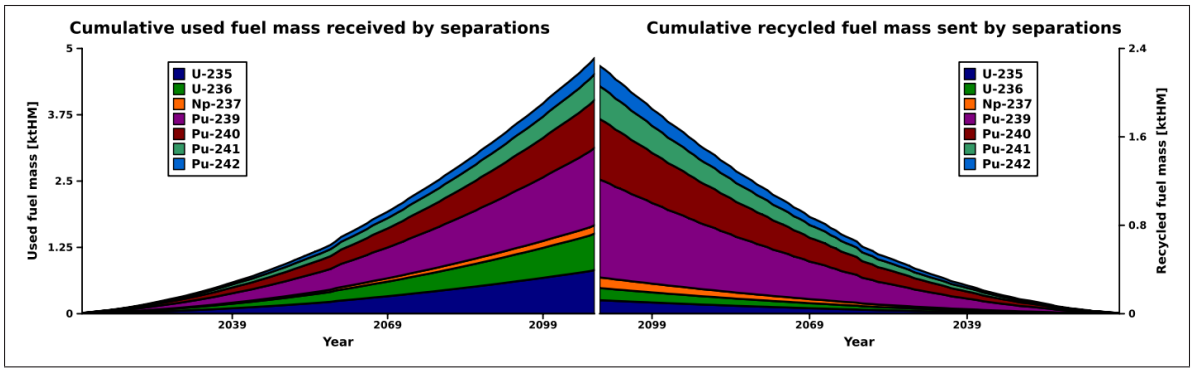

Figure 5.11 Mass flows by selected isotope both into and out of the separations plant in Problem 9. Note the different y-axis scales between the left- and right-hand images and the reversing of the time axis for the material sent *from* separations to facilitate comparisons of the two flows' compositions. The within-stream proportions are roughly constant, but the between-stream proportions show the cumulative effect of the approximation scheme favoring the plutonium stream over the uranium stream.

behave quite similarly to their VISION counterparts. Problems 5 and 6 demonstrated the code's affinity-based customer and supplier matching capabilities for fuel cycle services (fabrication), and Problems 7 and 8 served an analogous role for matching of fuel cycle material supply and demand (unenriched uranium). All of these matching problems demonstrate the somewhat arbitrary and unpredictable routing effects caused by degeneracy in the NFP formulations, a behavior expected to matter less and less as more subtle arc-costing schemes eliminate much of the degeneracy.

Specific to closed fuel cycles, the recipe approximation formulations were subjected to unit tests (Tests 1-7) and a full scenario featuring a thermal MOX recycle (Problem 9). Together tests 6 and 7 showed that a recipe approximation constraint on neutronics is necessary to obtain the desired behavior in certain situations but can do more harm than good in others. Such was indeed the case in Problem 9, where the best approximations were obtained by removing the neutronics constraint altogether and allowing the algorithm to match purely on isotopic and total mass. Results from these early recipe approximation studies show that much work remains to be done, both in the formulation itself and in tuning separations plant operation to ensure that formulation has suitable material to choose from.

Chapter 6

Summary and future work

6.1 Summary

This thesis has reported on the state of GENIUSv2 at the point in its development when it had reached an important milestone: the ability to run large multi-region scenarios that include all of the basic nuclear fuel cycle steps, including reprocessing. The work this document has described has helped lay the foundation for the goal of investigating the advanced-fuel-cycle-related research questions discussed in Chapter 1. This work includes the design and implementation of the discrete-facilities/discrete-materials fuel cycle model described in Section 3.1, the simulation machinery and robust input-output infrastructure described throughout the rest of Chapter 3, and—most importantly—Chapter 4's preliminary methodology for using optimization techniques to determine appropriate material routing instructions and recipe approximation choices. The fuel cycle modeling problems reported in Chapter 5 are intended to demonstrate and characterize the current behavior of the code and to point out areas for future improvement.

To our knowledge, GENIUSv2 is the first code to adapt linear and network flow programming techniques for the purposes of global optimization of nuclear fuel cycles. The complexity of the fuel cycle system—especially its many commodities, the infungibility of some of those commodities, and the different purposes and modes of operation of each of the types of nuclear fuel cycle facility—makes this a challenging task. At times, these challenges have required the adoption of an air of pragmatism—most notably through the introduction of physical and mathematical approximations—in order to arrive at a code that can produce meaningful results

and to which more complex and sophisticated techniques can be added in the future. The naive way GENIUS currently forms NFPs to solve the routing problem on a month-by-month basis and the poor quality of the neutronic weighting constraint in the recipe approximation algorithm are probably the two most notable examples, but there are others as well. Most of these areas for future work were identified as they became relevant, but to close they are reviewed briefly below.

6.2 Future work

6.2.1 Facility data and behavior

The task labelled fuel cycle "tuning" throughout this document is the most obvious and probably straightforward opportunity for immediately improving the quality of GENIUSv2 modeling. In essence, this task involves identifying—through some combination of research into industry practice and trial and error—appropriate kinds of facility behavior that benefit the functioning of the fuel cycle as a whole (and parameters that quantify this behavior). For instance, it's clearly in no one's best interest for facilities to order material only at the exact time they "realize" they need it. But identifying appropriate mechanisms for forecasting supply and demand and maintaining appropriately sized buffers of material will require careful thought, especially because such decisions represent local heuristics that could interfere with the search for globally optimal material routing strategies.

Another set of modeling difficulties related to facility behavior involves the issue of fuel cycle commodities' varying fungibility. Recall, for instance, the way that the source and size of the material objects processed and stored in the separations plant determines both the size of the LP GENIUS solves during recipe approximation and the variety of the "wine cellar" from which it chooses appropriate materials for constructing recycled fuel. In Problem 9, that variety caused great variability in the quality of the recipe approximations, an effect that should be eliminated in the future.

Finally, developers may wish to begin examining possible methods for including more explicit and detailed economic modeling in GENIUS, either as a post-processing step or as

runtime functionality that can affect the flow of material and the status of facilities or their institutional owners. Past CNERG intern Arnaud Reveillere worked on early implementations of the post-processing-based approach, and Jain and Wilson have proposed paradigms for treating facilities' cash flows and financial outlooks explicitly (2006). Obviously, allowing economic considerations to affect the flow of material may require adjusting or replacing the network-based MRP formulations currently in place.

6.2.2 Material routing problem

Most of the other future work called for by those formulations was discussed in Sections 4.2 and 4.3 and relates to the following four issues: discrete/continuous tension, degeneracy, arc costing, and the time horizon. The first problem arises for those commodities where the manager does not wish to "split" a customer's request between two different suppliers as if it were a continuous quantity. For instance, unlike in the case of yellowcake or unenriched uranium, it's hard to imagine a reactor ever wanting a batch of fuel it ordered to be provided by two separate suppliers. Unfortunately, because fabricators make their offers with respect to the amount of mass they're able to process, requests for fuel batches are converted to an equivalent mass as well, and there's nothing to guarantee that all of the mass selected to flow to the sink representing that request will come from a single source node (fabricator). One promising approach that would not cause the class of optimization problem to become significantly harder would involve figuring out how to teach the fabricators to make reliable offers based on a number of batches instead of a mass. If both the supply and demand for fabricated fuel batches are expressed in terms of *integer* divergences and constraints (like a number of batches), then the solution would be guaranteed to include only integer flows as well, provided the code used a true network solver[1].

[1] This is a surprising and convenient property of linear network flow programs with integer constraints. See Bertsekas (1998)

The problems of degeneracy and arc costing are related; the greater the variation in the arc costs of the network associated with a particular commodity, the fewer the number of degenerate optimal solutions. So it seems that the most expedient means of eliminating problematic degeneracies is to increase the level of detail in the process of assigning arc costs. As more real-world data is incorporated into in the model, mechanisms for choosing arc costs that better represent the true competition in the system will likely present themselves. Of course, the goal of identifying a form whose objective function explicitly minimizes the cost of producing electricity (rather than simply the costs of matching as much supply to demand as possible) may help determine whether exploring this arc-costing research question is worth the time and effort.

Finally, it should be a high priority to identify a more appropriate (i.e., longer) time horizon for solving the routing problem (in whatever form) and to modifying the other aspects of the code to accommodate this change, as necessary. The operations research and energy economics literature will likely be of great help in doing so, as might some of the underlying methodologies implemented in the MARKAL code and other more rigorous platforms for modeling energy economics.

6.2.3 Recipe approximation problem

The two biggest concerns with the current RAP formulations are (1) the challenges posed by expressing rigorous neutronics constraints in a form consistent with common linear programming techniques and (2) operating the chemical separations schemes themselves in such a way that the algorithm itself has appropriately composed candidate barrels to choose from. Some of the pressure on both these tasks could be relieved with the incorporation of a simplified burnup engine such as the one developed by Scopatz and Schneider, since its use would at least allow the code to capture the consequences of failing to provide a closely matched recipe. In the case of the consistently under-reactive approximations in Problem 9, it would allow us to calculate a (consequently smaller) maximum burnup for the recycled fuel, in addition to correcting the subsequent miscalculation of the recycled fuel's end-of-life isotopics (without

this capability, error in the input recipe isotopics becomes even larger error in the output recipe isotopics).

6.2.4 Fuel cycle design problem

Finally, we return (for completeness' sake) to the question of the fuel cycle design problem as a whole. The decision to require that the exact facility deployment be specified as input rather than being determined dynamically according to runtime heuristics serves as an important example of the techniques that will be necessary in working toward the goal of making GENIUS compatible with "wrapper-based" optimization techniques like simulated annealing, genetic algorithms, etc. Of course, in addition to this work within the code, a very large area of future GENIUS-related work will be to design the actual strategy for linking the code to appropriate optimization software. A useful first step might be a literature review of the available open-source choices; DAKOTA (Eldred et al., 2008), a robust toolkit maintained at Sandia National Laboratories for use on design optimization problems like this one, seems like a strong contender. In any event, this work will be important to ensure that GENIUSv2 continue toward its fourth and most challenging design principle: to be a generative tool for fuel cycle analysis *and* design.

Bibliography

Allali, A., Bojariu, R., Diaz, S., Elgizouli, I., Griggs, D., Hawkins, D., Hohmeyer, O., Jallow, B. P., Kajfez-Bogataj, L., Leary, N., Lee, H., and Wratt, D. (2007). Climate change 2007: Synthesis report. Technical report, Intergovernmental Panel on Climate Change. Accessed 5 January 2009 from `http://www.ipcc.ch/pdf/assessment-report/ar4/syr/ar4_syr.pdf`.

Allison, G. (2005). *Nuclear Terrorism: The Ultimate Preventable Catastrophe*, chapter Through the Prism of 9/11, pages 123–139. Henry Hold and Company, New York.

Benedict, M. and Pigford, T. (1981). *Nuclear Chemical Engineering*. McGraw-Hill Publishing Co., Columbus, OH, second edition.

Bertsekas, D. P. (1998). *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Nashua, NH.

Board on Energy and Environmental Systems (2008). Review of DOE's nuclear energy research and development program. Technical report, National Research Council, Washington, DC. Accessed 5 January 2009 from `http://www.nap.edu/catalog/11998.html`.

Boscher, T., Romano, A., Hejzlar, P., Kazimi, M., and Todreas, N. (2004). The CAFCA code for simulation of nuclear fuel cycles: Description of methodology, assumptions, and initial results. Technical Report Report No. MIT-NFC-TR-069, Massachusetts Institute of Technology, Cambridge, MA.

Chinneck, J. W. (2007). *Practical Optimization: A Gentle Introduction*, chapter Network Flow Programming, pages 1–12. [Self-published], Ottawa, ON.

Cochran, R. G. and Tsoulfanidis, N. (1990). *The Nuclear Fuel Cycle: Analysis and Management*. American Nuclear Society, La Grange Park, IL, second edition.

COIN-OR (2007). CLP 1.7 [software]. `https://projects.coin-or.org/Clp`.

Darst, R. G. and Dawson, J. I. (2008). Baptists and bootleggers, once removed: The politics of radioactive waste internalization in the European Union. *Global Environmental Politics*, 8(2):17–38.

de la Garza, A. (1977). Uranium-236 in light water reactor spent fuel recycled to an enriching plant. *Nuclear Technology*, 32:176–185.

Doman, L. E., Staub, J., Mayne, L., Barden, J., Martin, P., Mellish, M., Kerney, D., Kette, S., Aniti, L., Murphy, B., Kapilow-Cohen, B., and Lindstrom, P. (2008). International energy outlook. Technical report, Energy Information Administration. Accessed 5 January 2009 from `http://www.eia.doe.gov/oiaf/ieo/pdf/0484(2008).pdf`.

Dunzik-Gougar, M. L., Juchau, C. A., Pasamehmetoglu, K., Wilson, P. P. H., Oliver, K. M., Turinsky, P. J., Abdel-Khalik, H. S., Hays, R., and Stover, T. E. (2007). Global Evaluation of Nuclear Infrastructure Utilization Scenarios (GENIUS). In *Global 2007: Advanced Nuclear Fuel Cycles and Systems*. American Nuclear Society.

Eldred, M. S., Adams, B. M., Haskell, K., Bohnhoff, W. J., Eddy, J. P., Gay, D. M., Hart, W. E., Hough, P. D., Kolda, T. G., Swiler, L. P., and Watson, J. P. (2008). *DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis*. Sandia National Laboratories, Albuquerque, NM, 4.2+ edition. Report No. SAND2006-6337.

Ferris, M. C., Mangasarian, O. L., and Wright, S. J. (2008). *Linear Programming with Matlab*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA, first edition.

Fischer, P., Kaushik, D., Nowak, D., Siegel, A., Yang, W. S., and Pieper, G. W. (2008). Advanced simulation for fast reactor analysis. Accessed 20 April 2009 from U.S. Department of Energy Office of Science, http://www.scidacreview.org/0803/html/nuclear.html.

Forrest, J., de la Nuez, D., and Lougee-Heimer, R. (2004). *CLP User Guide*. IBM. Accessed 10 May 2009 from http://www.coin-or.org/Clp/userguide/clpuserguide.html.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Upper Saddle River, NJ, illustrated edition.

GEN IV International Forum (2009). Generations of nuclear energy. Accessed 20 April 2009 from OECD Nuclear Energy Agency, http://www.gen-4.org/Technology/evolution.htm.

Grady, R. M. (2008). Development of economic accounting for nuclear waste in fuel cycle analysis. Master's thesis, University of Wisconsin-Madison.

Gray, P., Hart, W., Painton, L., Phillips, C., Trahan, M., and Wagner, J. (1997). A survey of global optimization methods. Technical report, Sandia National Laboratories, Albuquerque, NM. Accessed May 1 2009 from http://www.cs.sandia.gov/opt/survey/main.html.

Hermann, O. W. and Westfall, R. M. (1998). ORIGEN-S: SCALE system module to calculate fuel deplettion, actinide transmutation, fission product buildup and decay, and associated radiation source terms. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN. Report No. NUREG/CR-0200.

ILOG (2008). CPLEX 11.0 [software]. http://www.ilog.com/products/cplex/.

isee systems (2008). STELLA 9 [software]. http://www.iseesystems.com/.

Jacobi, W. M. (1989). Fast breeder reactors for energy security. *Nuclear Technology*, 88(2):183–189.

Jacobson, J., Yacout, A., Matthern, G., Piet, S., Shropshire, D., and Laws, C. (2007). VISION 2: Enhanced simulation model of the next generation nuclear fuel cycle. In *Transactions of the American Nuclear Society*, volume 96, pages 199–200.

Jacobson, J., Yacout, A. M., Matthern, G., Piet, S., Shropshire, D., and Laws, C. (2006). VISION: Verifiable fuel cycle simulation model. In *Transactions of the American Nuclear Society*, volume 95, pages 157–9.

Jain, R. and Wilson, P. P. H. (2006). Transitioning to global optimization in fuel cycle system study tools. In *Transactions of the American Nuclear Society*, volume 95, pages 162–3.

Juchau, C. (2008). Development of the global evaluation of nuclear infrastructure and utilization scenarios (GENIUS) nuclear fuel cycle systems analysis code. Master's thesis, Idaho State University.

Juchau, C. A. and Dunzik-Gougar, M. L. (2006). A review of nuclear fuel cycle systems codes. Technical report, SINEMA LDRD Project. Accessed 13 February 2007 from `http://thesinema.org/`.

Juchau, C. A., Dunzik-Gougar, M. L., and Pasamehmetoglu, K. (2006). Simulation Institute for Nuclear Energy Modeling and Analyses (SINEMA): Developing a GENIUS. In *Transactions of the American Nuclear Society*, volume 94, pages 78–9.

Kelly, J. and Savage, C. (2005). Advanced Fuel Cycle Initiative program plan. Technical report, Advanced Fuel Cycle Initiative, Washington, DC. Accessed 5 January 2009 from `http://afci.sandia.gov/downloads/2005_AFCI_Program_Plan.pdf`.

Lippman, S. B. (1991). *C++ Primer*. Addison-Wesley, Upper Saddle River, NJ, 2nd edition.

Lisowski, P. (2007). Global Nuclear Energy Partnership. In *Global Nuclear Energy Partnership Annual Meeting*, Litchfield Park, AZ. Global Nuclear Energy Partnership.

Loulou, R., Goldstein, G., and Noble, K. (2004). *Documentation for the MARKAL Family of Models*. Energy Technology Systems Analysis Programme. Accessed 23 April 2009 from http://www.etsap.org/MrklDoc-I_StdMARKAL.pdf.

McCarthy, K. (2007). Systems analysis campaign. In *Global Nuclear Energy Partnership Annual Meeting*, Litchfield Park, AZ. Global Nuclear Energy Partnership.

Miron, A. (2008). Identification and analysis of critical gaps in nuclear fuel cycle codes required by the SINEMA program. Technical Report Project Number 07-071, University of Cincinnati, Cincinatti, OH. Accessed 23 April 2009 from http://www.ne.doe.gov/neri/2007Awards/NERI07-071Abstract.pdf.

Nuclear and Radiation Studies Board (2008). Internationalization of the nuclear fuel cycle: Goals, strategies, and challenges [prepublication copy]. Technical report, National Academy of Sciences, National Research Council, and Russian Acadmy of Sciences, Washington, DC. Accessed 6 January 2009 from http://www.nap.edu/catalog/12477.html.

Nuclear Regulatory Commission (2009). Expected new nuclear power plant applications. Accessed 20 April 2009 from Nuclear Regulatory Commission, http://tinyurl.com/mo9tfh.

Nuttall, W. J. (2005). *Nuclear Renaissance: Technologies and Policies for the Future of Nuclear Power*. CRC Press, Boca Raton.

Nystrom, I. and Wene, C.-O. (1999). Energy-economy linking in MARKAL-MACRO: Interplay of nuclear, conservation and CO2 policies in Sweden. *International Journal of Environment and Pollution*, 12(2/3):323–342.

Oliver, K. M., Elmore, R., Haney, K., and Huff, K. (2008). *genius-2.0 Documentation*. UW-Madison Computational Nuclear Engineering Research Group, Madison, WI, 2.0 edition. Accessed April 30 2009 from http://cnerg.engr.wisc.edu/GENIUS2/docs/.

Oliver, K. M. and Fatenejad, M. (2009). Object-oriented programming [C++ boot camp course notes]. Technical report, The Hacker Within, UW-Madison, Madison, WI. Accessed 29 April 2009 from `http://hackerwithin.org/cgi-bin/hackerwithin.fcgi/wiki/CppBootCampOop`.

Oliver, K. M., Wilson, P. P. H., Reveillere, A., Ahn, T. W., Dunn, K., Huff, K., and Elmore, R. (2009). Studying international fuel cycle robustness with the GENIUSv2 discrete facilities/materials fuel cycle systems analysis tool [in press, paper accepted]. In *Global 2009: The Nuclear Fuel Cycle: Sustainable Options & Industrial Perspectives*. French Nuclear Energy Society.

Pasamehmetoglu, K. O. and Finck, P. (2006). SINEMA: Simulation Institute for Nuclear Energy Modeling and Analyses. In *Transactions of the American Nuclear Society*, volume 95, pages 155–6.

Phillips, A., Jacobson, J., and Shropshire, D. (2007). VISION.ECON: A dynamic model for estimating nuclear fuel cycle costs. In *Transactions of the American Nuclear Society*, volume 96, page 121.

Pidd, M. (2003). *Tools for Thinking: Modelling in Management Science*. John Wiley and Sons, Hoboken, NJ.

Piet, S. (2007). Updated conceptualization of the “winery” issue. Personal Communication.

Powersim Inc. (2006). *Tutorial on How to Use the SimCoupler Module*. Accessed 9 June 2009 from `www.psim-europe.com/openload2.php?doc=tutorialSimcoupler.pdf`.

Powersim Software (2008). Studio 7 [software]. `http://www.powersim.com`.

Python Software Foundation (2006). Python 2.5 [software]. `http://www.python.org/`.

Radel, T. E. (2007). Repository modeling for fuel cycle scenario analysis. Master’s thesis, University of Wisconsin-Madison.

Sauter, R. and Awerbuch, S. (2003). Oil price volatility and economic activity: A survey and literature review. Technical report, International Energy Agency, Paris. Accessed 5 January 2009 from `http://tinyurl.com/m8sqs4`.

Scopatz, A. M. and Schneider, E. A. (2009). A new method for rapid computation of transient fuel cycle material balances [in press]. *Nuclear Engineering and Design*, doi:10.1016/j.nucengdes.2009.02.022:1–16.

Shropshire, D. (2007). Advanced fuel cycle cost basis. Technical Report DE-AC07-05ID14517, Idaho National Laboratory.

Sims, R. E., Rogner, H.-H., and Gregory, K. (2003). Carbon emission and mitigation cost comparisons between fossil fuel, nuclear and renewable energy resources for electricity generation. *Energy Policy*, 31:1315–1326.

SQLite Consortium (2008). SQLite [software]. Available at `http://www.sqlite.org/`.

Van Den Durpel, L., Yacout, A., Wade, D., and Khalil, H. (2003). DANESS dynamic analysis of nuclear system strategies. In *Global 2003: Atoms for Prosperity: Updating Eisenhower's Global Vision for Nuclear Energy*, pages 1613–1620, New Orleans, LA. American Nuclear Society.

Van Den Durpel, L., Yacout, A. M., and Wade, D. C. (2007). Status on developments and applications of the integrated nuclear energy system code DANESS. In *Transactions of the American Nuclear Society*, volume 96, pages 212–14.

Vidal, C. J. and Goetschalckx, M. (1997). Strategic production-distribution models: A critical review with emphasis on global supply chain models. *European Journal of Operational Research*, 98:1–18.

Voorspools, K. R., Brouwers, E. A., and D'haeseleer, W. D. (2000). Energy content and indirect greenhouse gas emissions embedded in 'emission-free' power plants: Results for the Low Countries. *Applied Energy*, 67:307–330.

Wigeland, R. A., Bauer, T. H., Fanning, T. H., and Morris, E. E. (2006). Separations and transmutation criteria to improve utilization of a geologic repository. *Nuclear Technology*, 154(1):95–106.

Wilson, P. P. H. and Oliver, K. M. (2007). Designing GENIUS Version 2 to model inter-facility relationships [poster]. In *Global Nuclear Energy Partnership Annual Meeting*, Litchfield Park, AZ. Global Nuclear Energy Partnership.

Yacout, A. M., Jacobson, J. J., Matthern, G., Piet, S. J., and Moisseytsev, A. (2006a). VISION – A dynamic model of the nuclear fuel cycle [preprint]. Accessed 1 May 2008 from `http://www.inl.gov/technicalpublications/Documents/3479816.pdf`.

Yacout, A. M., Jacobson, J. J., Matthern, G., Piet, S. J., Shropshire, D. E., and Laws, C. (2006b). VISION – Verifiable fuel cycle simulation of nuclear fuel cycle dynamics [preprint]. Accessed 26 May 2009 from `http://www.inl.gov/technicalpublications/Documents/3394908.pdf`.

Yi, S. K. (2008). Nuclear fuel cycle modeling approaches for recycling and transmutation of spent nuclear fuel. Master's thesis, The Ohio State University.

DISCARD THIS PAGE

Appendix A: Useful terms from object-oriented programming

A.1 Classes, objects, and inheritance

Inheritance is a feature of object-oriented programming that allows for the creation of increasingly specialized *objects*. A common illustration of inheritance is via the creation of a taxonomy of classes representing living things (see Lippman, 1991; Oliver and Fatenejad, 2009). In object-oriented programming, a *class* is the blueprint for creating individual objects or *instances,* so a programmer who wants to model animals will first write an animal class. This class will define the state and behavior possible for animals in general. Next, he or she can write specialized *subclasses* that *derive* or *inherit from* the animal class. Instances of these subclasses (say, an object representing a bear or a rabbit) then inherit the data and abilities of the *superclass* from which they are derived—in addition to whatever bear- or rabbit-specific data and abilities are defined in their respective classes.

A.2 Member data and methods

Member data are declared in a class and store the state of its objects. For instance, the data member representing the age of an animal will obviously be different for different instances that are different ages. Similarly, the *methods* or *member functions* of a class determine each object's behavior. These functions are invoked on particular instances, likely changing their state in some way. Member functions of a superclass can be either inherited or *overridden* by its subclasses.

A.3 Static data and methods

Static data and methods are not associated with particular instantiations of a class but with the class as a whole. For instance, if there were a limit on the population of a particular animal species, it might be enforced via static data that tracked the number of instantiations of that

particular animal subclass and a static function that routinely checked that number and could take appropriate action when it got too high.

A.4 Singletons

Described by Gamma et al. (1994), a *singleton* class object is specifically designed to be universally accessible to any object that knows about it, via a static function of the singleton class. For this behavior to work, we must require that only one instance of the singleton class is ever created.

Appendix B: Input and output file tables

B.1 Input

Five tables are currently needed to form a valid GENIUSv2 input file: `Regions`, `Insts`, `Facs`, `FacParams`, and `Rules`. These tables, included in an SQLite database that gets passed in to the code as a command-line argument, are described below.

Table B.1 Description of `Regions` table input.

Column	Data type	Description
`regID`	INTEGER PRIMARY KEY	A unique identifier for this region.
`name`	TEXT	A name for this region.
`type`	TEXT	An enumeration specifying whether this region operates fuel cycle facilities other than reactors.
`demand`	BLOB	A specially formatted block of memory that encodes the monthly electricity demand for this region.

Table B.2 Description of `Insts` table input.

Column	Data type	Description
`instID`	INTEGER PRIMARY KEY	A unique identifier for this institution.
`regID`	INTEGER	The identifier for the region where this institution is located.
`name`	TEXT	A name for this institution.
`build`	BLOB	A specially formatted block of memory that encodes this institution's plan for building future facilities.

Table B.3 Description of `Facs` table input.

Column	Data type	Description
`facID`	INTEGER PRIMARY KEY	A unique identifier for this facility.
`instID`	INTEGER	The identifier for the institution that owns this facility.
`name`	TEXT	A name for this facility.
`yearStartOp`	INTEGER	The year this facility started (or will start) operating.
`monthStartOp`	INTEGER	The month this facility started (or will start) operating.
`constrTime`	INTEGER	The number of months it takes (or took) to construct this facility.
`lifeTime`	INTEGER	The number of months this facility will operate.
`cycleTime`	INTEGER	The number of months it takes to perform a cycle of this facility's characteristic operation.
`status`	TEXT	An enumeration specifying the operational status of this facility at the beginning of the simulation (operating, under construction, etc.).
`capFactor`	FLOAT	A typical capacity factor for this facility.
`capacity`	FLOAT	An appropriate measure of this facility's capacity (units vary).
`type`	TEXT	An enumeration specifying what kind of facility this is.
`batchesPer-Core`	INTEGER	The number of fuel batches this facility uses in its core, if it is a reactor.
`feed`	TEXT	The commodity or commodities this facility uses as feedstock(s).
`prod`	TEXT	The commodity or commodities this facility produces.
`tailsFrac`	FLOAT	The default tails fraction used by this facility, if it performs enrichment.
`oreWF`	FLOAT	The weight fraction of uranium present in the yellow-cake produced by this facility, if it is a mine.
`freshRec`	INTEGER	The recipe this facility uses for fresh fuel, if it is a reactor.
`spentRec`	INTEGER	The recipe this facility produces as spent fuel, if it is a reactor.

Table B.4 Description of `FacParams` table input.

Column	Data type	Description
`ID`	INTEGER PRIMARY KEY	A unique identifier for this class of generic future facilities.
`type`	TEXT	An enumeration specifying what kind of facility the members of this generic future type are.
`name`	TEXT	A name for this generic facility type.
`lifeTime`	INTEGER	The number of months facilities of this type will operate.
`constrTime`	INTEGER	The number of months it takes to construct a facility of this type.
`cycleTime`	INTEGER	The number of months it takes to perform a cycle of this facility type's characteristic operation.
`charCF`	FLOAT	A characteristic capacity factor for this generic facility type.
`capacity`	FLOAT	An appropriate measure of this facility type's capacity (units vary).
`batchesPer-Core`	INTEGER	The number of fuel batches members of this facility use in their cores, if they are reactors.
`feed`	TEXT	The commodity or commodities this facility type uses as feedstock(s).
`prod`	TEXT	The commodity or commodities this facility type produces.
`tailsFrac`	FLOAT	The default tails fraction used by this facility type, if it performs enrichment.
`oreWF`	FLOAT	The weight fraction of uranium present in the yellow-cake produced by this facility type, if it is a mine type.
`freshRec`	INTEGER	The recipe this facility type uses for fresh fuel, if it is a reactor type.
`spentRec`	INTEGER	The recipe this facility type produces as spent fuel, if it is a reactor type.

Table B.5 Description of `Rules` table input.

Column	Data type	Description
`fromType`	TEXT	The actor type of the supplier (`Region`, `Inst`, or `Fac`) involved in this rule.
`fromID`	INTEGER	The identifier of the supplier involved in this rule.
`toType`	TEXT	The actor type of the customer (`Region`, `Inst`, or `Fac`) involved in this rule.
`toID`	INTEGER	The identifier of the customer involved in this rule.
`commodity`	TEXT	The commodity type this rule applies to.
`affinity`	FLOAT	The affinity for trade this rule describes.
`timeStart`	INTEGER	The time at which this rule begins to apply.
`timeEnd`	INTEGER	The time at which this rule ceases to apply.

B.2 Output

Three tables currently have GENIUSv2 output written to them during or at the end of the simulation: `Facs`, `MatFacHist`, and `MatIsoHist`. `Facs` data is written as additional columns in the table that already exists; `MatFacHist` and `MatIsoHist` are entirely new tables not present in the original input file. This output is described below.

Table B.6 Description of `Facs` table output.

Column	Data type	Description
`startOp`	INTEGER	The GENIUS time at which this facility began operating.
`capLog`	BLOB	A specially formatted block of memory that encodes the monthly capacity factors this facility actually operated with during the simulation.

Table B.7 Description of `MatFacHist` table output.

Column	Data type	Description
`matID`	INTEGER	The unique identifier of the material object whose transfer is stored in this record.
`time`	INTEGER	The time at which this material transfer occurred.
`fromFac`	INTEGER	The identifier of the facility that sent the material.
`toFac`	INTEGER	The identifier of the facility that received the material.
`compID`	INTEGER	The identifier of the composition of the material at the time of the transfer.

Table B.8 Description of `MatIsoHist` table output.

Column	Data type	Description
`compID`	INTEGER PRIMARY KEY	The identifier of the material composition stored in this record.
`time`	INTEGER	The time at which this composition applied to some material.
`comp`	BLOB	A specially formatted block of memory that encodes the isotopic composition stored in this record.