Numerical Experiments for Verifying Demand Driven Deployment Algorithms

Draft 2

Jin Whan Bae and Gwendolyn Chee

2017-10-11

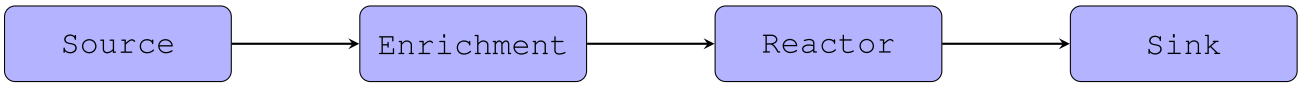
1 Introduction

The Demand-Driven Cycamore Archetype project (NEUP-FY16-10512) aims to develop CYCAMORE demand-driven deployment capabilities. The project plans to use non-optimizing, deterministic-optimizing and stochastic-optimizing prediction algorithms.

These prediction models are being developed by the University of South Carolina. In this report, we discuss numerical experiments for testing the non-optimizing, deterministic optimizing and stochastic optimizing methods. The numerical experiments will be designed for both the once through nuclear fuel cycle and advanced fuel cycles.

2 Once through Nuclear Fuel Cycle

Figure 1: Flow Chart of Once through Nuclear Fuel Cycle



In this report, the once-through fuel cycle is simplified to have 4 components: `Source`, `Enrichment`, `Reactor` and `Sink`. However, in a true once-through fuel cycle, there are more than 4 components. Other components include milling, conversion, fuel fabrication etc. For example, if a fuel cycle includes the Fuel Fabrication step and excludes the `Enrichment` step, the tests can be modified to include Fuel Fabrication and exclude `Enrichment`.

2.1 Input File Specification

CYCLUS uses XML input files. For the input file, we assume the archetype to be an `INSTITUTION`, since it governs deployment and decommission of facilities.

The user would only have to define the reactor prototype and reactor deployment. The remaining fuel facilities, both front end and back end, would be recognized by the archetype by looking at each prototypes' in-and-out commodities. Then the recognized, or 'connected' fuel cycle facilities deploy with demand. If an input file does not have the necessary 'connections' for the reactor to receive fuel, it will throw an error.

We suggest an example format: First, the input file would define reactor prototype to be deployed. There can be multiple reactors.

```
<institution>
      <name>NO_ddd</name>
       <config><NO_ddd/></config>
      <reactor_list>
        <val>lwr</val>
        <val>sfr</val>
        <val>mox_lwr</val>
      </reactor_list>
```

Then deployment of the reactors is defined, and the user is given the option to manually define the deployment(DeployInst):

```
      <deployment>
      <type>manual</type>

        <build_times>
          <val>1</val>
          <val>10</val>
          <val>20</val>
          <val>40</val>
        </build_times>
        <n_build>
          <val>3</val>
          <val>3</val>
          <val>3</val>
          <val>3</val>
        </n_build>
        <lifetimes>
          <val>960</val>
          <val>960</val>
          <val>960</val>
          <val>960</val>
        </lifetimes>
      </deployment>

    </institution>
```

Or have the institution deploy reactors according to power demand (GrowthRegion):

```
      <deployment>
      <type>growth</type>

      <growth>
        <piecewise_function>
              <piece>
                <start>0</start>
                <function>
                  <type>linear</type>
                  <params>1 2</params>
                </function>
              </piece>
        </piecewise_function>
      </growth>

      </deployment>

    </institution>
```

2.2 Simple Scenario for Testing

A simple scenario definition is given where a predefined `Reactor` is deployed at timestep one and three. The chain of front—end fuel cycle facility of one facility each (`Source`, `Enrichment`, `FuelFab`) can supply enough fuel for the operation of one reactor. A `Sink` can hold enough amount for two batches of fuel. The specifics are listed below:

Simulation Parameters	Value	Units
Duration	7	timesteps
Deploy Reactors	1, 3	timesteps
Natural U Composition	0.71% U235	
Tails Assay	0.003	% U235
Fuel Enrichment	4	% U235
Enrichment throughput	300	$\frac{kg}{month \cdot facility}$
Enrichment SWU Capacity	2000	$\frac{SWU}{month \cdot facility}$
Source Throughput	3000	$\frac{kg}{month \cdot facility}$
Sink Capacity	200	$\frac{kg}{facility}$

Table 1: Simulation Parameters

Reactor Parameters	Value	Units
Cycle Time	2	timesteps
Refuel Time	1	timesteps
Lifetime	6	timesteps
Power Capacity	1000	MWe
Assembly Size	100	kg
# assemblies per core	3	
# assemblies per batch	1	

Table 2: Reactor Parameters

An example input file with the facility and simulation definitions can be found in the input directory of this repository.

2.3 Analytical Solution

The simple scenario is solve analytically.

Timestep	Fuel [kg]	Enrichment $[\frac{SWU}{month}]$	Source $[\frac{kg}{month}]$	Sink Capacity [kg]	Note
1	300	1583	2701	0	Reactor1 core load
2	0	1583	2701	0	
3	400	2111	3601	100	Reactor2 core load, Reactor1 refuel
4	0	2111	3601	0	
5	100	528	901	200	Reactor2 refuel
6	0	528	901	500	Reactor1 decom
7	0	528	901	500	
8	0	528	901	800	Reactor2 decom
9	0	528	901	800	

Table 3: Analytical Solution

Timestep	Deploy Enrichment	Deploy Source	Deploy Sink
1	1	1	0
2	0	0	0
3	1	0	1
4	0	1	0
5	0	0	0
6	0	0	2
7	0	0	0
8	0	0	1
9	0	0	0

Table 4: Deployment Needs

Time Step	Total Power Output [MWe]
1	1000
2	1000
3	1000
4	2000
5	1000
6	2000
7	1000
8	1000
9	0
10	0

Table 5: Total Power Output

2.4 Non-optimizing prediction method

The following conditions need to be satisfied for each segment of the fuel cycle. The method to how each condition is tested is described. Tests written in C++ for each condition is included in Appendix A.

2.4.1 `Reactor`

1. *Do all the `Reactors` run at full capacity (not lacking fuel)?*
   This is tested by determining the total number of time steps which both reactors are in operation, one reactor is in operation or none is in operation. Based on the values of the analytical solution in Table 5, there should be 2 time steps where both reactors are in operation, 6 time steps where only one reactor is in operation and 2 time steps where no reactors are in operation.

2. *Is a new `Reactor` deployed when the energy demand exceeds the energy produced by the current `Reactors`?*
   This is tested by comparing the total number of reactors deployed to the analytical solution. Based on Table 4, in total 2 reactors were deployed.

3. *Is a `Reactor` decommissioned when the energy demand falls behind the output of the current `Reactors`?*

2.4.2 `Enrichment`

1. Is the enriched uranium produced by `Enrichment` within the upper limit of [insert uncertainty] for the analytic solution of the enriched uranium required by the `Reactor` for each time step?

2. Is a new `Enrichment` facility deployed when the enriched uranium required by the `Fuelfab` exceeds the enriched uranium produced of current `Enrichment` facilities?

3. Is a `Enrichment` decommissioned when the enriched uranium required by the `Fuelfab` falls behind the enriched uranium produced by current `Enrichment` facilities?

2.4.3 `Source`

1. Is the mined uranium produced by `Source` within the upper limit of [insert uncertainty] for the analytic solution of the mined uranium required by the `Enrichment` facilities for each time step?

2. Does the `Source` mined uranium produced increase when mined uranium required by the `Enrichment` facilities exceeds mined uranium produced by the current `Source`?

3. Does the `Source` mined uranium produced decrease when mined uranium required by the `Enrichment` facilities falls behind the mined uranium required by the current `Source`?

2.5 Deterministic-Optimizing/Stochastic prediction method

The following conditions need to be satisfied for each segment of the fuel cycle.

1. Do all the `Reactors` run?

```
TEST(ReactorTests, DDDeploy_DO) {
    [Example input with the following attributes:]
        [int simdur = 20;]
        [Defines reactor with zero refueling cycle and operation
        cycle of 1 month]
        [Defines fuel cycle facilities parameters]
        [Defines Reactor Deploy Scheme / Power Demand]
        [Increasing Fuel Demand with Time]
    [Run test]
    [Test if Reactor has no zero values in output Timeseriespower]
}
```

2. Do all the `Reactors` run at full capacity (not lacking fuel)?

```
TEST(ReactorTests, DDDeploy_DO) {
    [Example input with the following attributes:]
        [int simdur = 20;]
        [Defines reactor with zero refueling cycle, power capacity
        of x and operation cycle of 1 month]
        [Defines fuel cycle facilities parameters]
        [Defines Reactor Deploy Scheme / Power Demand]
        [Increasing Fuel Demand with Time]
    [Run test]
    [Test if Reactor has x value in output Timeseriespower]
}
```

3. Is the objective function optimized?

4. Is the constraint followed?

5. Do the related fuel cycle facilities get deployed upon demand?

```
TEST(ReactorTests, DDDeploy_NO) {
    [Example input with the following attributes:]
        [int simdur = 20;]
        [Defines reactor with zero refueling cycle and operation
        cycle of 1 month]
        [Defines fuel cycle facilities parameters]
        [Defines Reactor Deploy Scheme / Power Demand]
        [Increasing Fuel Demand with Time]
    [Run test]
    [Test if fuel facility is deployed in the beginning]
    [Test if fuel facility is deployed later in the simulation
    (have analytic solution)]
}
```

6. Do the related fuel cycle facilities exit upon demand decrease?

```
TEST(ReactorTests, DDDeploy_NO) {
    [Example input with the following attributes:]
        [int simdur = 20;]
        [Defines reactor with zero refueling cycle and operation
        cycle of 1 month]
        [Defines fuel cycle facilities parameters]
        [Defines Reactor Deploy Scheme / Power Demand]
        [Decreasing Fuel Demand with Time]
    [Run test]
    [Test if fuel facility is deployed in the beginning]
    [Test if fuel facility exits later in the simulation (have
    analytic solution)]
}
```

3 Advanced Fuel Cycles

4 Appendix A - Sample Test Code

4.1 Reactor

1. *Do all the* `Reactors` *run at full capacity (not lacking fuel)?*

```
TEST(ReactorTests, FullCapacity) {
   std::string config =
     "  <fuel_inrecipes>  <val>fresh_uox</val>  </fuel_inrecipes>  "
     "  <fuel_outrecipes> <val>spent_uox</val>  </fuel_outrecipes>  "
     "  <fuel_incommods>  <val>uox</val> </fuel_incommods>  "
     "  <fuel_outcommods> <val>spent_uox</val>      </fuel_outcommods>
     "  <fuel_prefs>      <val>1.0</val>        </fuel_prefs>  "
     ""
     "  <cycle_time>2</cycle_time>  "
     "  <refuel_time>1</refuel_time>  "
     "  <assem_size>100</assem_size>  "
     "  <n_assem_core>3</n_assem_core>  "
     "  <n_assem_batch>1</n_assem_batch>  ";

  int simdur = 10
  cyclus::MockSim sim(cyclus::AgentSpec(":cycamore:Reactor"),config
  ,simdur);
sim.AddSource("uox").Finalize();
sim.AddRecipe("fresh_uox",c_uox());
sim.AddRecipe("spent_uox".c_spentuox());
int id. = sim.Run();

int both_on = 2;
std::vector<Cond> conds;
conds.push_back(Cond("Value", "==",2000));
QueryResult qr = sim.db().Query("TimeSeriesPower", &conds);
EXPECT_EQ(both_on,qr.rows.size());

int one_on = 6;
std::vector<Cond> conds;
conds.push_back(Cond("Value", "==",1000));
QueryResult qr = sim.db().Query("TimeSeriesPower", &conds);
EXPECT_EQ(one_on,qr.rows.size());

int none_on = 2;
std::vector<Cond> conds;
conds.push_back(Cond("Value", "==",0));
QueryResult qr = sim.db().Query("TimeSeriesPower", &conds);
EXPECT_EQ(one_on,qr.rows.size());
}
```

2. *Is a new* `Reactor` *deployed when the energy demand exceeds the energy produced by the current* `Reactors`*?*

```
TEST(ReactorTests, DeployNew) {
   std::string config =
     "  <fuel_inrecipes>  <val>fresh_uox</val>  </fuel_inrecipes>  "
     "  <fuel_outrecipes> <val>spent_uox</val>  </fuel_outrecipes>  "
     "  <fuel_incommods>  <val>uox</val> </fuel_incommods>  "
     "  <fuel_outcommods> <val>spent_uox</val>      </fuel_outcommods>
     "  <fuel_prefs>      <val>1.0</val>        </fuel_prefs>  "
     ""
     "  <cycle_time>2</cycle_time>  "
     "  <refuel_time>1</refuel_time>  "
   "  <assem_size>100</assem_size>  "
   "  <n_assem_core>3</n_assem_core>  "
   "  <n_assem_batch>1</n_assem_batch>  ";

int simdur = 10
cyclus::MockSim sim(cyclus::AgentSpec(":cycamore:Reactor"),config,
simdur);
sim.AddSource("uox").Finalize();
sim.AddRecipe("fresh_uox",c_uox());
sim.AddRecipe("spent_uox".c_spentuox());
int id. = sim.Run();

int reactors_deployed = 2;
std::vector<Cond> conds;
conds.push_back(Cond("String", "==",:cycamore:Reactor));
QueryResult qr = sim.db().Query("AgentEntry", &conds);
EXPECT_EQ(reactors_deployed,qr.rows.size());
}
```