Numerical Experiments for Verifying Demand Driven Deployment Algorithms

Jin Whan Bae and Gwendolyn Chee

2017-10-11

---

1 Introduction

For many fuel cycle simulations, it is currently up to the user to define a deploy scheme, or facility parameters, to make sure that there's no gap in the supply chain. Or, the same goal is achieved by setting the `facility` capacity to infinity, which does not reflect real-world conditions.

The Demand-Driven Cycamore Archetype project (NEUP-FY16-10512) aims to develop CYCAMORE demand-driven deployment capabilities. The developed algorithm, in the form of CYCLUS `Institution` agent, deploys `facilities` to meet the front-end and back-end demand of the fuel cycle.

This report provides numerical tests for 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 Method

This report lists necessary capabilities of the new CYCLUS `institute` for demand-driven deployment of fuel cycle facilities. Then the report lists tests to check correct implementation of the capabilities, with a sample fuel cycle with well-defined facility parameters.

3 Configuration

The user defines prototypes to be deployed for fuel facilities, and reactor deployment scheme. The reactor deployment causes the demand of fuel which triggers fuel facility deployment. The detailed input file XML input schema is shown in Appendix A.

4 Logic Flow

The algorithm creates a supply chain with the fuel facilities, then calculates the demand for each facility. A simple example is illustrated in figure 1.
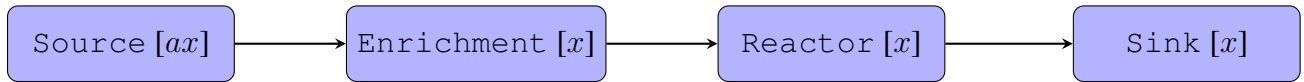
```
Source [ax]  →  Enrichment [x]  →  Reactor [x]  →  Sink [x]
```

Figure 1: Simple flow of materials. The values in the bracket are demands calculated by the algorithm. The `Reactor` *demands x amount of fuel, which translates into demands of x from* `Enrichment` *and* `Sink`*. The* `Source` *has a demand of ax to take in for enrichment losses. $a \approx 9$ in LEU PWRs.*

If the demand of a commodity exceeds the current supply, the `Institution` builds more facilities to meet the demand for that commodity.

4.1 Reactor Parameters for Test Scenarios

Table 1 provides the parameters for the `reactors` in the test scenarios and Table 2 provides the parameters for the `source`, `enrichment` and `sink` facilities in the test scenarios.

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 1: Reactor Parameters

Parameters	Value	Units
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 2: Source, Enrichment and Sink Facility Parameters

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

4.2 Non-optimizing prediction method

To ensure that the non-optimizing prediction model is working correctly, each function of the code must be tested to determine if its output matches the analytical solution. In this section, the tests that must be met is described based on the parameters defined in Table 1 and 2 and analytical solution of a defined simple scenario. Unit test examples are included in Appendix B.

Test 1: A reactor outputs the user defined power capacity when deployed and does not output power capacity when decommissioned.

Test Scenario: A single reactor is deployed at time step 3 and decommissioned at time step 5.

Analytical Solution: During cycle time, the reactor will have a power output of 1000MWe. Therefore, there is only be power output of 1000MWe at time steps 3 and 4. The analytical solution of the power output of the reactor per time step is given in Table 3.

Timestep	Reactor Power Output (MWe)
1	0
2	0
3	1000
4	1000
5	0
6	0

Table 3: Analytical solution of the power output of the reactor per time step

Test 2: A reactor outputs the user defined power capacity during its cycle time and not during its refuel time.

Test Scenario: A single reactor is deployed at time step 3 and decommissioned at time step 8.

Analytical Solution: Based on the reactor parameters defined in Table 1, the reactor will have a cycle time of 2 time steps and a refuel time of 1 time step. Therefore, there is only be power output of 1000MWe during cycle time steps. The analytical solution of the power output of the reactor per time step is given in Table 4.

Timestep	Reactor Power Output (MWe)
1	0
2	0
3	1000
4	1000
5	0
6	1000
7	1000
8	0
9	0

Table 4: Analytical solution of the power output of the reactor per time step

Test 3:1 or more of the facility that produces reactor's input commodity (fuel) is deployed when the amount of fuel available is below the amount of fuel required by the reactor.
Test Scenario: A single reactor is deployed at time step 3.
Analytical Solution: Based on the simple supply chain example in Figure 1 and enrichment facility parameters defined in Table 2, an enrichment facility must be deployed at time step 3 to meet fuel demand. The analytical solution of the fuel available per time step is given in Table 5.

Timestep	Fuel (kg)
1	0
2	0
3	300
4	0

Table 5: Analytical solution of the fuel available per time step

5   Appendix A - parameter configuration

First, the user defines the prototypes to be deployed to support the reactor.

```
<interleave>
  <element name="institution">
    <data type="string"/>
  </element>
    <element name="prototypes">
      <oneOrMore>
        <element name="val">
          <data type="string"/>
        </element>
      <oneOrMore>
    </element>
</interleave>
```

Code 1: Fuel cycle facility prototype definition

Then deployment of the power-producing reactors is defined, and the user is given two choices:

- Manually define the deployment scheme of reactors (code 2)

- Define power demand equation (code 3)

```
<interleave>
    <element name="reactors">
        <oneOrMore>
            <element name="val">
                <data type="string"/>
            </element>
        </oneOrMore>
    </element>
    <element name="build_times">
        <oneOrMore>
            <element name="val">
                <data type="int"/>
            </element>
        </oneOrMore>
    </element>
    <element name="n_build">
        <oneOrMore>
            <element name="val">
                <data type="int"/>
            </element>
        </oneOrMore>
    </element>
    <optional>
        <element name="lifetimes">
            <oneOrMore>
                <element name="val">
                    <data type="int"/>
                </element>
            </oneOrMore>
        </element>
    </optional>
</interleave>
```

Code 2: Manual definition of reactor deployment

```xml
<interleave>
    <element name="growth">
        <oneOrMore>
            <element name="item">
                <interleave>
                    <element name="reactors">
                        <data type="string"/>
                    </element>
                    <element name="piecewise_function">
                        <oneOrMore>
                            <element name="piece">
                                <interleave>
                                    <element name="start">
                                        <data type="int"/>
                                    </element>
                                    <element name="function">
                                        <interleave>
                                            <element name="type">
                                                <data type="string"/>
                                            </element>
                                            <element name="params">
                                                <data type="string"/>
                                            </element>
                                        </interleave>
                                    </element>
                                </interleave>
                            </element>
                        </oneOrMore>
                    </element>
                </interleave>
            </element>
        </oneOrMore>
    </element>
</interleave>
```

Code 3: Definition of power demand function

6 Appendix B - Sample Test Code

6.1 Reactor

Condition 1: All the reactors run at full capacity

```
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());
  }
```

Condition 2: A new `Reactor` is deployed when the energy demand exceeds the energy pro-duced 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;
```

```cpp
conds.push_back(Cond("String", "==",:cycamore:Reactor));
QueryResult qr = sim.db().Query("AgentEntry", &conds);
EXPECT_EQ(reactors_deployed,qr.rows.size());
}
```