Examination – November 2, 2022

CS 787 – Decision Guidance Systems – Professor Alex Brodsky

Consider the posted folder **cs787_ha3_and_exam_sn_template** folder. Make sure that Pyomo package is installed:
https://pyomo.readthedocs.io/en/stable/index.html

The final exam is an extension of HA3, including an extension of the analytic model **am** in the module **ams.py** (in the **solution** sub-folder) for general hierarchical service networks. Examples of input and output of are given under the **example_input_output** folder

Problem 1

Instructions

Under the subfolder **solution,** duplicate the Python module **ams_extended_template.py** into **ams_extended.py**
Implement the required analytic models (see below) by filling out **ams_extended.py** template. Example inputs and corresponding outputs are given in the folder **example_input_output.** To run the implemented functions in Problem 1 below, use **solution/main_extended.py**

a. In terminal, make the downloaded folder your current folder
b. Run **main_extended.py** in Python w/ stdin being the model input stdout being the model output.

Example:

python solution/main_extended.py < example_input_output/sn_extended_in.json > answers/out.json

to get answers in **out.json** which you can compare with the correct answer, e.g., in **example_input_output/sn_extended_out.json**

Questions

Make sure that the following analytic models are implemented as Python functions in the module **ams_extended.py** under the sub-folder **solution**
You may use your solutions from HA3 and/or update them.

1. **flowBoundConstraint(flowBounds, flow)**
    a. example **flowBounds**: see in **sn_extended_in.json** under inFlow or outFlow of services
    b. example **flow:** see in **sn_extended_out.json** under inFlow or outFlow of services
    c. this function now handles optional upper bounds "ub" in addition to "lb" in inFlow or outFlow of input (like **sn_extended_in.json)**

d. this function returns true if, for all flow ids f in flow
   i. qty >= 0
   ii. if there's a lower bound for f in flowBounds, qty >= lb
   iii. if there's an upper bound for f in flowBounds, qty <= ub

   and returns false otherwise.

e. Note: must use dgal.all for aggregating constraints (Boolean and)

2. supplierMetrics(input)

a. example input: see in **sn_extended_in.json** under "sup1" or "sup2" service

b. example output: see in **sn_extended_out.json** under "sup1" or "sup2" service

c. **cost** in the output structure is the total cost of procurement from the supplier in the **input** based on **ppu** (price-per-unit) and purchased **qty.**

d. **co2** in the output structure is the total co2 emissions associated with procurement from the supplier based on **co2pu** (co2-per-unit) and qty of sold items

e. **constraints** in the output structure involve **boundConstraints** using the function flowBoundConstrains, and **activeServiceConstraint** – given in the template – which is based on **totalQtyOut** – total qty in outFlow of the supplier

3. manufMetrics(input)

a. example input: see in **sn_extended_in.json** under "manuf1" or "manuf2" service

b. example output: see in **sn_extended_out.json** under "manuf1" or "manuf2" service

c. **cost** in the output structure is the total cost of manufacturer in the **input** computed based on **ppu** (price-per-unit) and ordered **qty,** for all ordered products, which are represented as keys in **outFlow**

d. **co2** in the output structure is the total co2 of manufacturer in the **input** computed based on **co2pu** (co2-per-unit) and ordered **qty,** for all ordered products, which are represented as keys in **outFlow**

e. **qty** of each key in **inFlow** is computed based on **qty**'s of ordered items in **input["outFlow"]** and on **input["qtyInPer1out"].** For example, if **input["outFlow"]["table"] = 100,** and **input["qtyInPer1out"]["table"]["table_leg"] = 4,** then **output["inFlow"]["table_leg"]** should be 100 * 4 = 400.

f. **constraints** in the output structure involve **boundConstraints** for inFlow and outFlow in the output using the function flowBoundConstrains, and **activeServiceConstraint** – given in the template – which is based on **totalQtyOut** – total qty in outFlow of the manufacturer

4. transportMetrics(input,shared)

a. example input: see in **sn_extended_in.json** under "transp1" or "transp2" service

b. example output: see in **sn_extended_out.json** under "transp1" or "transp2" service

c. **input["orders"]** contains transportation orders. Each order is described using (**in, out, sender, recipient, qty**) where **in** and **out** indicate item/loc being sent and received; **sender** and **recipient** are business entities (described in **shared**), which have an associated location.

d. **Cost** in the output is computed as follows. For every pair of (source, destination) locations, compute the total weight of all shipments (in **orders**). Then, the cost for a (source, destination) is total weight times price-per-lb (in **input["pplbFromTo"])** Finally, the total cost is aggregation of cost for all (source, destination) pairs.
e. **Co2** in the output is computed similar to cost as follows. For every pair of (source, destination) locations, compute the total weight of all shipments (in **orders**). Then, the co2 for a (source, destination) is total weight times co2perlb (in **input["co2perlbFromTo"])** Finally, the total co2 is aggregation of cost for all (source, destination) pairs.
f. **inFlow** and **outFlow** quantities in the output, for every item/loc key, is computed from the corresponding orders
g. **constraints** in the output structure involve **boundConstraints** for inFlow and outFlow in the output using the function flowBoundConstrains, and **activeServiceConstraint** – given in the template – which is based on **totalOrderQty** – the total qty shipped in all orders

5. **am(input)** by implementing the function **compute_metrics(shared,root,services)** for general service network, extended as follows.
   a. example input: **sn_extended_in.json**
   b. example output: **sn_ sn_extended_out.json**
   c. **shared, root, services** have the structure of the corresponding parts in **input** to the function **am** (see input example)
   d. **cost** in **compute_metrics** is computed by aggregating costs over all **subServices** of the **root** service
   e. **co2** in **compute_metrics** is computed by aggregating co2 over all **subServices** of the **root** service
   f. **inOutFlowKeysSet** is the set union of inFlow and outFlow keys of the root service
   g. **flowKeysSet** is the set of all flows from inFlow and outFlow of the root service, as well as from inFlow and outFlow of all subServices of the root service
   h. **supply** in the computation of **subServicesFlowSupply,** for flow **f** in **flowKeysSet**, is the total quantity of flow **f** coming from all **subServices** in their **outFlow.**
   i. **demand** in the computation of **subServicesFlowDemand,** for flow **f** in **flowKeysSet**, is the total quantity of flow **f** coming into all **subServices** in their **inFlow**
   j. **qty** in the computation of **newInFlow,** for flow **f** in input **inFlow,** is the difference between the **demand** and **supply** of flow **f**
   k. **qty** in the computation of **newOutFlow,** for flow **f** in input **outFlow,** is the difference between the **supply** and **demand** of flow **f**
   l. the constraint computed (to replace placeholder True in the template) for flow **f** must reflect the balance between **supply** and **demand** of flow **f**
   m. the overall constraints in the return structure are the conjunction of the constraints **internalSupplySatisfiesDemand, inFlowConstraints, outFlowConstraints, subServiceConstraints** and new constraints:

i. **activeServiceConstraint -** given in the template and based on the number of active sub-services (i.e., with on flag being 1) in var **noActiveSubServices** that needs to be computed
ii. **activeSubServicesBound** that hat is True if and only if
    1. there's no "maxActiveSubServices" key under root service, or
    2. the number of active sub-services is bounded by maxActiveSubServices (value corresponding to this key under the root service)

Problem 2

Questions

Run the following:

1. To optimize the service network, with input captured in **example_input_output/sn_extended_in_var.json,**
2. run: **python solution/optSN_extended.py**

You can compare the results of your optimization with the results given under the folder **answers_correct**

Optionally, play with constants in constraints and in the model input, such as prices, so that you can predict the optimal values that optimization should choose. See if this is indeed the case. If not, try to debug the model etc.

To submit

Upload the file **ams_extended.py**
Nothing else!